

Oracle® GoldenGate

Oracle GoldenGate for Distributed Applications and Analytics



Release 26ai
G49812-04
May 2026

ORACLE®

Copyright © 2015, 2026, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience	i
Documentation Accessibility	i
Conventions	i
Related Information	ii

1 Overview

1.1 Understanding Oracle GoldenGate for Distributed Applications and Analytics	1
1.1.1 Understanding Oracle GoldenGate for Distributed Applications and Analytics	1
1.1.1.1 Delivery Configuration Options	1
1.1.1.2 Adapter Integration Options	2
1.1.1.3 Monitoring Performance	4
1.2 What's Supported in Oracle GoldenGate for Distributed Applications and Analytics	4
1.2.1 Verifying Certification and System Requirements	4
1.2.2 Understanding Handler Compatibility	5
1.2.3 What are the Additional Support Considerations?	5
1.3 Dependency Downloader	6
1.3.1 Dependency Downloader Setup	6
1.3.2 Running the Dependency Downloader Scripts	7
1.3.3 Dependency Downloader Scripts	8

2 Prepare

2.1 Preparing for Installation	1
2.1.1 Downloading Oracle GoldenGate for Distributed Applications and Analytics	1
2.1.2 Installation Overview	2
2.1.2.1 Contents of the Installation ZIP File	2
2.1.2.2 Using the Generic Build of Oracle GoldenGate	3
2.1.2.3 Considerations for Using a Custom Build for a GoldenGate for Distributed Applications and Analytics Instance of Oracle GoldenGate	3
2.1.2.4 Installing to a Non-Generic Instance of Oracle GoldenGate	3
2.1.3 Directories and Variables in Microservices Architecture	4
2.1.4 Setting up Environmental Variables	6

2.1.4.1	Java on Linux/UNIX	7
2.1.4.2	Java on Windows	7

3 Install

3.1	Setting up Oracle GoldenGate for Distributed Applications and Analytics in a High Availability Environment	1
3.1.1	Running GG for DAA from a Single Instance	1
3.1.2	Running GG for DAA on a Cluster of Servers	1
3.1.3	Shared Storage	2
3.2	Installing Oracle GoldenGate for Distributed Applications and Analytics	2
3.2.1	Installing Oracle GoldenGate MA for Distributed Applications and Analytics Using the UI	3
3.2.2	Silent Installation	4
3.2.3	Setting Up Secure or Non-Secure Deployments	4
3.2.3.1	How to Add Secure or Non-Secure Deployments	5
3.2.3.2	How to Remove a Deployment	10

4 Get Started

4.1	Getting Started with Oracle GoldenGate for Distributed Applications and Analytics	1
4.1.1	Working With Deployments	1
4.1.2	About Oracle GoldenGate Properties Files	2
4.1.2.1	Parameter Files	2
4.1.3	Using the Admin Client	2
4.1.4	Controlling Oracle GoldenGate (Microservices Architecture) Processes	2

5 Upgrade

5.1	Upgrading Oracle GoldenGate for Distributed Applications and Analytics	1
5.1.1	Obtaining the Oracle GoldenGate Distribution	1
5.1.2	Scope of Upgrade	2
5.1.2.1	Replicat Upgrade Considerations	2
5.1.3	Upgrading Oracle GoldenGate for Distributed Applications and Analytics – GUI Based	2

6 Secure

6.1	Security	1

7 Configure

7.1	Configuring Oracle GoldenGate for Distributed Applications and Analytics	1

7.1.1	Running with Replicat	1
7.1.1.1	Replicat Grouping	2
7.1.1.2	About Replicat Checkpointing	2
7.1.1.3	About Initial Load Support	2
7.1.1.4	About the Unsupported Replicat Features	2
7.1.1.5	How the Mapping Functionality Works	2
7.1.2	About Schema Evolution and Metadata Change Events	2
7.1.3	About Configuration Property CDATA[] Wrapping	3
7.1.4	Using Regular Expression Search and Replace	3
7.1.4.1	Using Schema Data Replace	4
7.1.4.2	Using Content Data Replace	4
7.1.5	Scaling Oracle GoldenGate for Distributed Applications and Analytics Delivery	5
7.1.6	Coordinated Apply Support	8
7.1.7	Configuring Cluster High Availability	10
7.1.8	Using Identities in Oracle GoldenGate Credential Store	11
7.1.8.1	Creating a Credential Store	11
7.1.8.2	Adding Users to a Credential Store	11
7.1.8.3	Configuring Properties to Access the Credential Store	12
7.2	Logging	13
7.2.1	About Replicat Process Logging	13
7.2.2	About Java Layer Logging	13
7.2.3	About SQL Statement Logging	14
7.2.3.1	Configuring SQL Statement Logging	15
7.3	Configuring Logging	15
7.3.1	Oracle GoldenGate Java Adapter Default Logging	15
7.3.1.1	Default Logging Setup	15
7.3.1.2	Log File Name	15
7.3.1.3	Changing Logging Level	16
7.3.2	Recommended Logging Settings	16
7.3.2.1	Changing to the Recommended Logging Type	16

8 Quickstarts

8.1	QuickStarts: Prerequisites	1
8.2	Realtime Data Ingestion into Snowflake with Oracle GoldenGate for Distributed Applications and Analytics	2
8.2.1	Prerequisites for Internal Staging	2
8.2.2	Install Dependency Files	3
8.2.3	Create a Credential Store Entry	3
8.2.4	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	4
8.3	Realtime Parquet Ingestion into Google Cloud Storage with GoldenGate for Distributed Applications and Analytics 23.8 and later	9

8.3.1	Prerequisites	9
8.3.2	Install Required Dependency Files	10
8.3.3	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	11
8.4	Realtime Parquet Ingestion into Google Cloud Storage with Oracle GoldenGate for Distributed Applications and Analytics	16
8.4.1	Prerequisites	16
8.4.2	Install Dependency Files	16
8.4.3	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	17
8.5	Realtime Parquet Ingestion into AWS S3 Buckets with Oracle GoldenGate for Distributed Applications and Analytics 23.8 and later	23
8.5.1	Prerequisites	23
8.5.2	Install Required Dependency Files	24
8.5.3	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	25
8.6	Realtime Parquet Ingestion into AWS S3 Buckets with Oracle GoldenGate for Distributed Applications and Analytics	31
8.6.1	Prerequisites	31
8.6.2	Install Dependency Files	31
8.6.3	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	32
8.7	Realtime Parquet Ingestion into Azure Data Lake Storage with Oracle GoldenGate for Distributed Applications and Analytics 23.8 and later	38
8.7.1	Prerequisites	38
8.7.2	Install Required Dependency Files	39
8.7.3	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	39
8.8	Realtime Parquet Ingestion into Azure Data Lake Storage with Oracle GoldenGate for Distributed Applications and Analytics	45
8.8.1	Prerequisites	45
8.8.2	Install Dependency Files	45
8.8.3	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	46
8.9	Realtime Parquet Ingestion into OCI Object Storage with Oracle GoldenGate for Distributed Applications and Analytics 23.8 and later	52
8.9.1	Prerequisites	52
8.9.2	Install Required Dependency Files	53
8.9.3	Configure Credentials for Oracle Cloud Infrastructure	54
8.9.4	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	54
8.10	Realtime Parquet Ingestion into OCI Object Storage with Oracle GoldenGate for Distributed Applications and Analytics	60
8.10.1	Prerequisites	60
8.10.2	Install Dependency Files	60

8.10.3	Configure Credentials for Oracle Cloud Infrastructure	62
8.10.4	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	62
8.11	Realtime Message Ingestion to OCI Streaming with Oracle GoldenGate for Distributed Applications and Analytics	68
8.11.1	Prerequisites	69
8.11.2	Install Dependency Files	69
8.11.3	Create Kafka Producer Properties	69
8.11.4	Create a Replicat in GG for DAA	70
8.12	Realtime Message Ingestion to Azure Event Hubs with Oracle GoldenGate for Distributed Applications and Analytics	75
8.12.1	Prerequisites	75
8.12.2	Install Dependency Files	75
8.12.3	Create a producer.properties for Azure Event Hubs	76
8.12.4	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	76
8.13	Realtime Data Ingestion into GCP BigQuery with Oracle GoldenGate for Distributed Applications and Analytics	82
8.13.1	Prerequisites for Google Cloud Platform BigQuery Stage and Merge	82
8.13.2	Install Dependency Files	82
8.13.3	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	83
8.14	Realtime Message Ingestion to Google Pub/Sub with Oracle GoldenGate for Distributed Applications and Analytics	89
8.14.1	Prerequisites	90
8.14.2	Install Dependency Files	90
8.14.3	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	90
8.15	Realtime Message Ingestion to Apache Kafka with Oracle GoldenGate for Distributed Applications and Analytics	95
8.15.1	Prerequisites	96
8.15.2	Install Dependency Files	96
8.15.3	Create Kafka Producer Properties File	96
8.15.4	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	97
8.16	Realtime Data Ingestion into Azure Databricks (unity catalog enabled) with GoldenGate for DAA	101
8.16.1	Prerequisites for Databricks Replication with Unity Catalog	102
8.16.2	Install Dependency Files	102
8.16.3	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	103
8.17	Realtime Data Ingestion into Oracle AI Data Platform with Oracle GoldenGate for DAA	110
8.17.1	Prerequisites	110
8.17.2	Install Required Dependency Files	110

8.17.3	Configure Credentials for Oracle Cloud Infrastructure	111
8.17.4	Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics	111

9 Best Practices

9.1	Apache Kafka	1
9.1.1	Configuration Considerations and Best Practices	1
9.1.2	Oracle GoldenGate Kafka Replicat Topic Mapping	2
9.1.2.1	Table-per-Topic Pattern	2
9.1.2.2	All-Tables-to-Single-Topic Pattern	3
9.1.3	Oracle GoldenGate Kafka Replicat Schema Registry and Schema Propagation	4
9.1.4	Oracle GoldenGate Kafka Replicat Partition Selection and Order Consistency	4
9.1.5	Propagating Additional Metadata with Oracle GoldenGate Kafka Replicat	5
9.1.6	Oracle GoldenGate Kafka Replicat Message Modes and Maintaining Transactional Boundaries	6
9.1.6.1	Operation Mode (OP)	6
9.1.6.2	Transaction Mode (TX)	6
9.1.7	Oracle GoldenGate Kafka Replicat Message Formatting	7
9.1.8	Security Considerations and Best Practices	10
9.1.8.1	Securing Oracle GoldenGate	10
9.1.8.2	Network Transport Security (Oracle GoldenGate to Kafka) and Client Authorization	10
9.1.9	Performance Considerations	11
9.1.9.1	Oracle GoldenGate Extract Performance	11
9.1.9.2	Oracle GoldenGate Kafka Replicat Performance	12
9.1.9.3	Network Latency	13
9.2	Cloud Object Storage Replication	13
9.2.1	Introduction	14
9.2.2	Architecture Overview	15
9.2.3	Configuration Considerations and Best Practices	16
9.2.3.1	File Rollover Configuration	16
9.2.3.2	Partitioning	18
9.2.4	DDL Changes and Schema Propagation	19
9.2.5	PK Updates	20
9.2.6	DATE and TIMESTAMP Types	21
9.2.7	Number Types	22
9.2.8	Performance Considerations	23
9.3	Snowflake	25
9.3.1	Introduction	25
9.3.2	Architecture Overview	26
9.3.3	Configuration Considerations and Best Practices	27
9.3.3.1	Snowflake Session and Connection Stability	27

9.3.3.2	Snowflake Authentication	28
9.3.3.3	Precise Instantiation for Snowflake	30
9.3.4	Snowflake Schema Change Handling	31
9.3.5	LOB Type Replication	32
9.3.6	Snowflake Apache Iceberg Tables	33
9.3.7	Handling PK Updates for Snowflake Stage and Merge Replicat	34
9.3.8	Oracle GoldenGate Snowflake Performance Considerations	35

10 Replicate Data

10.1	Source	1
10.1.13	Setting Up and Running the Kinesis Streams Handler	2
10.1.13.1	Set the Classpath in Kinesis Streams Handler	3
10.1.13.2	Kinesis Streams Handler Configuration	3
10.1.13.3	Using Templates to Resolve the Stream Name and Partition Name	9
10.1.13.4	Resolving AWS Credentials	10
10.1.13.5	Configuring the Proxy Server for Kinesis Streams Handler	11
10.1.13.6	Configuring Security in Kinesis Streams Handler	12
10.1.1	Add Extract	12
10.1.2	Amazon MSK	13
10.1.3	Amazon DocumentDB	13
10.1.4	Apache Cassandra	13
10.1.4.1	Overview	14
10.1.4.2	Setting Up Cassandra Extract Change Data Capture	14
10.1.4.3	Deduplication	17
10.1.4.4	Topology Changes	17
10.1.4.5	Data Availability in the CDC Logs	17
10.1.4.6	Using Initial Load Extract	18
10.1.4.7	Using Change Data Capture Extract	18
10.1.4.8	Replicating to RDMBS Targets	20
10.1.4.9	Partition Update or Insert of Static Columns	21
10.1.4.10	Partition Delete	21
10.1.4.11	Security and Authentication	21
10.1.4.12	Cleanup of CDC Commit Log Files	23
10.1.4.13	Multiple Extract Support	28
10.1.4.14	CDC Configuration Reference	29
10.1.4.15	Troubleshooting	46
10.1.4.16	Cassandra Capture Client Dependencies	49
10.1.5	Apache Kafka	50
10.1.5.1	Overview	50
10.1.5.2	Prerequisites	50
10.1.5.3	General Terms and Functionality of Kafka Capture	51

10.1.5.4	Generic Mutation Builder	56
10.1.5.5	Kafka Connect Mutation Builder	57
10.1.5.6	Example Configuration Files	60
10.1.6	Azure Event Hubs	61
10.1.7	Confluent Kafka	61
10.1.8	DataStax	61
10.1.9	Java Message Service (JMS)	61
10.1.9.1	Prerequisites	61
10.1.9.2	Oracle GoldenGate Java Delivery	62
10.1.9.3	Configuring Message Capture	62
10.1.10	MongoDB	65
10.1.10.1	Overview	66
10.1.10.2	Prerequisites to Setting up MongoDB	66
10.1.10.3	MongoDB Database Operations	68
10.1.10.4	Using Extract Initial Load	68
10.1.10.5	Using Change Data Capture Extract	69
10.1.10.6	Positioning the Extract	69
10.1.10.7	Security and Authentication	70
10.1.10.8	MongoDB Bidirectional Replication	74
10.1.10.9	Mongo DB Configuration Reference	77
10.1.10.10	Columns in Trail File	86
10.1.10.11	Update Operation Behavior	88
10.1.10.12	Oplog Size Recommendations	89
10.1.10.13	Troubleshooting	90
10.1.10.14	MongoDB Capture Client Dependencies	90
10.1.11	Microsoft Fabric Eventstreams	91
10.1.11.1	Overview	92
10.1.11.2	Prerequisites	92
10.1.12	OCI Streaming	92
10.2	Target	93
10.2.1	Amazon DocumentDB	95
10.2.2	Add Replicat	95
10.2.3	Amazon Kinesis	97
10.2.3.1	Overview	97
10.2.3.2	Detailed Functionality	97
10.2.3.3	Kinesis Handler Performance Considerations	98
10.2.3.4	Troubleshooting	100
10.2.4	Amazon MSK	100
10.2.5	Amazon Redshift	101
10.2.5.1	Detailed Functionality	101
10.2.5.2	Operation Aggregation	101
10.2.5.3	Unsupported Operations and Limitations	102

10.2.5.4	Uncompressed UPDATE records	102
10.2.5.5	Error During the Data Load Proces	103
10.2.5.6	Troubleshooting and Diagnostics	103
10.2.5.7	Classpath	104
10.2.5.8	Configuration	104
10.2.5.9	INSERTALLRECORDS Support	115
10.2.5.10	Redshift COPY SQL Authorization	115
10.2.5.11	Co-ordinated Apply Support	117
10.2.5.12	Support for Mixed Case Identifiers	117
10.2.5.13	AWS Secrets Manager Authentication	117
10.2.5.14	Restrictions on Database Object Names	117
10.2.6	Amazon S3	118
10.2.6.1	Overview	118
10.2.6.2	Detailing Functionality	118
10.2.6.3	Configuring the S3 Event Handler	121
10.2.6.4	Enabling the Amazon S3 Multipart Upload	125
10.2.7	Apache Cassandra	126
10.2.7.1	Overview	127
10.2.7.2	Detailing the Functionality	127
10.2.7.3	Setting Up and Running the Cassandra Handler	132
10.2.7.4	About Automated DDL Handling	138
10.2.7.5	Performance Considerations	139
10.2.7.6	Additional Considerations	139
10.2.7.7	Troubleshooting	140
10.2.7.8	Cassandra Handler Client Dependencies	141
10.2.8	Apache HBase	143
10.2.8.1	Overview	143
10.2.8.2	Detailed Functionality	143
10.2.8.3	Setting Up and Running the HBase Handler	144
10.2.8.4	Security	148
10.2.8.5	Metadata Change Events	148
10.2.8.6	Additional Considerations	148
10.2.8.7	Troubleshooting the HBase Handler	149
10.2.8.8	HBase Handler Client Dependencies	150
10.2.9	Apache HDFS	162
10.2.9.1	Overview	162
10.2.9.2	Writing into HDFS in SequenceFile Format	162
10.2.9.3	Setting Up and Running the HDFS Handler	164
10.2.9.4	Writing in HDFS in Avro Object Container File Format	170
10.2.9.5	Generating HDFS File Names Using Template Strings	170
10.2.9.6	Metadata Change Events	171
10.2.9.7	Partitioning	171

10.2.9.8	HDFS Additional Considerations	173
10.2.9.9	Best Practices	174
10.2.9.10	Troubleshooting the HDFS Handler	174
10.2.9.11	HDFS Handler Client Dependencies	175
10.2.10	Apache Kafka	192
10.2.10.1	Overview	192
10.2.10.2	Apache Kafka Connect Handler	209
10.2.10.3	Apache Kafka REST Proxy	235
10.2.11	Apache Hive	248
10.2.12	Apache Iceberg	249
10.2.12.1	Detailed Functionality	249
10.2.12.2	Configuration	253
10.2.12.3	Configuration Templates	288
10.2.12.4	Limitations	288
10.2.12.5	Instantiating Oracle GoldenGate with an Initial Load	289
10.2.12.6	Iceberg Version 3 Support	292
10.2.12.7	Troubleshooting and Diagnostics	293
10.2.13	Azure Blob Storage	296
10.2.13.1	Overview	296
10.2.13.2	Prerequisites	296
10.2.13.3	Storage Account, Container, and Objects	296
10.2.13.4	Configuration	296
10.2.13.5	Troubleshooting and Diagnostics	301
10.2.14	Azure Data Lake Storage	302
10.2.14.1	Azure Data Lake Gen1 (ADLS Gen1)	302
10.2.14.2	Azure Data Lake Gen2 using Hadoop Client and ABFS	303
10.2.14.3	Azure Data Lake Gen2 using BLOB endpoint	306
10.2.15	Azure Event Hubs	306
10.2.16	Azure Synapse Analytics Data Warehouse	307
10.2.16.1	Detailed Functionality	307
10.2.16.2	Operation Aggregation	308
10.2.16.3	Compressed Update Handling	309
10.2.16.4	Configuration	309
10.2.16.5	Restrictions on Database Object Names	327
10.2.16.6	Troubleshooting and Diagnostics	327
10.2.17	Databricks	328
10.2.17.1	Detailed Functionality	328
10.2.17.2	Configuration	329
10.2.17.3	Troubleshooting and Diagnostics	354
10.2.17.4	Restrictions on Database Object Names	356
10.2.18	DataStax	356
10.2.19	Confluent Kafka	357

10.2.20	Elasticsearch	357
10.2.20.1	Elasticsearch with Elasticsearch 7x and 6x	357
10.2.20.2	Elasticsearch 8x	373
10.2.20.3	Support for Vector Data	383
10.2.21	Flat Files	384
10.2.21.1	File Writer Handler	384
10.2.21.2	Optimized Row Columnar (ORC)	399
10.2.21.3	Parquet	404
10.2.22	Google BigQuery	409
10.2.22.1	BigQuery Streaming Handler	409
10.2.22.2	Google BigQuery Stage and Merge	422
10.2.23	Google Cloud Storage	444
10.2.23.1	Overview	445
10.2.23.2	Prerequisites	445
10.2.23.3	Buckets and Objects	445
10.2.23.4	Authentication and Authorization	445
10.2.23.5	Configuration	447
10.2.23.6	Troubleshooting and Diagnostics	459
10.2.24	Google Cloud Managed Service for Apache Kafka	459
10.2.24.1	Overview	460
10.2.24.2	Configuration from Oracle Cloud Infrastructure (OCI)	460
10.2.24.3	Sample Configuration	461
10.2.25	Google Pub/Sub	461
10.2.25.1	Overview	461
10.2.25.2	Detailed Functionality	462
10.2.25.3	Setting up and Running the Google PubSub Handler	462
10.2.25.4	Configuring Handler Authentication	462
10.2.25.5	Google PubSub Handler Configuration	463
10.2.25.6	Proxy Settings	465
10.2.25.7	Sample Configuration	465
10.2.25.8	Google PubSub Dependencies	465
10.2.26	Java Message Service (JMS)	466
10.2.26.1	Overview	467
10.2.26.2	Setting Up and Running the JMS Handler	467
10.2.26.3	JMS Dependencies	474
10.2.27	Java Database Connectivity	475
10.2.27.1	Overview	475
10.2.27.2	Detailed Functionality	475
10.2.27.3	Setting Up and Running the JDBC Handler	477
10.2.27.4	Sample Configurations	480
10.2.28	Microsoft Fabric Eventstreams	482
10.2.28.1	Overview	483

10.2.28.2	Prerequisites	483
10.2.29	Microsoft Fabric OneLake	483
10.2.29.1	OneLake Event Handler Prerequisites	484
10.2.29.2	OneLake Mappings to Azure Data Lake Gen2	484
10.2.29.3	OneLake Event Handler Configuration	484
10.2.29.4	OneLake Event Handler Primary Key Update	491
10.2.29.5	Ingesting into Fabric Mirrored Tables using Partitioning	492
10.2.29.6	Managing the partnerEvents.json File for Fabric Mirrored Databases	493
10.2.29.7	OneLake Event Handler Troubleshooting and Diagnostics	494
10.2.30	MongoDB	497
10.2.30.1	Overview	497
10.2.30.2	MongoDB Wire Protocol	497
10.2.30.3	Supported Target Types	497
10.2.30.4	Detailed Functionality	498
10.2.30.5	Setting Up and Running the MongoDB Handler	499
10.2.30.6	Security and Authentication	502
10.2.30.7	Reviewing Sample Configurations	508
10.2.30.8	MongoDB to AJD/ATP Migration	509
10.2.30.9	Configuring an Initial Synchronization of Extract for a MongoDB Source Database using Precise Instantiation	511
10.2.30.10	Delivery to Oracle AI Database JSON Collection Tables	514
10.2.30.11	MongoDB Handler Client Dependencies	521
10.2.31	Oracle AI Data Platform	522
10.2.31.1	Detailed Functionality	523
10.2.31.2	Configuration	523
10.2.31.3	Troubleshooting and Diagnostics	530
10.2.31.4	Restrictions on Database Object Names	531
10.2.32	OCI Streaming	532
10.2.33	Oracle NoSQL	535
10.2.33.1	Overview	535
10.2.33.2	On-Premise Connectivity	536
10.2.33.3	OCI Cloud Connectivity	537
10.2.33.4	Oracle NoSQL Types	538
10.2.33.5	Oracle NoSQL Handler Configuration	539
10.2.33.6	Performance Considerations	541
10.2.33.7	Operation Processing Support	542
10.2.33.8	Column Processing	542
10.2.33.9	Table Check and Reconciliation Process	543
10.2.33.10	Oracle NoSQL SDK Dependencies	544
10.2.34	OCI Autonomous Data Warehouse	544
10.2.34.1	Detailed Functionality	545
10.2.34.2	ADW Database Credential to Access OCI ObjectStore File	545

10.2.34.3	ADW Database User Privileges	545
10.2.34.4	Unsupported Operations/ Limitations	545
10.2.34.5	Troubleshooting and Diagnostics	546
10.2.34.6	Classpath	549
10.2.34.7	Configuration	549
10.2.34.8	Restrictions on Database Object Names	555
10.2.35	OCI Object Storage	555
10.2.35.1	Overview	555
10.2.35.2	Detailing the Functionality	555
10.2.35.3	Configuration	556
10.2.35.4	Configuring Credentials for Oracle Cloud Infrastructure	561
10.2.35.5	Troubleshooting	562
10.2.35.6	OCI Dependencies	563
10.2.36	Redis	565
10.2.36.1	Data Structures Supported by the Redis Handler	566
10.2.36.2	Redis Handler Configuration Properties	569
10.2.36.3	Security	573
10.2.36.4	Authentication Using Credentials	574
10.2.36.5	SSL Basic Auth	574
10.2.36.6	SSL Mutual Auth	574
10.2.36.7	Redis Handler Dependencies	575
10.2.36.8	Redis Handler Client Dependencies	575
10.2.37	Snowflake	575
10.2.37.1	Snowflake Stage and Merge Handler	576
10.2.37.2	Snowflake Streaming Handler	601
10.2.37.3	Supported Snowflake Date Datatypes	607
10.2.37.4	Snowflake Iceberg Tables	607
10.2.38	Additional Details	609
10.2.38.1	HDFS Event Handler	609
10.2.38.2	Metacolumn Keywords	611
10.2.38.3	Metadata Providers	614
10.2.38.4	Pluggable Formatters	639
10.2.38.5	Stage and Merge Data Warehouse Replication	728
10.2.38.6	Template Keywords	734
10.2.38.7	Velocity Dependencies	739
10.2.39	Oracle GoldenGate Java Delivery	740
10.2.39.1	Configuring Java Delivery	740
10.2.39.2	Running Java Delivery	743
10.2.39.3	Configuring Event Handlers	745
10.2.39.4	Java Delivery Properties	747
10.2.39.5	Developing Custom Filters, Formatters, and Handlers	762

11 Administer

11.1	Automatic Heartbeat for Oracle GoldenGate for Distributed Applications and Analytics	1
11.1.1	Overview	1
11.1.2	Automatic Heartbeat Tables	3
11.1.2.1	ADD HEARTBEATTABLE	3
11.1.2.2	ALTER HEARTBEAT TABLE	4
11.1.2.3	INFO HEARTBEATTABLE	4
11.1.2.4	LAG	4
11.1.2.5	DELETE HEARTBEATTABLE	5
11.2	Parsing the Message	5
11.2.1	Parsing Overview	5
11.2.1.1	Parser Types	5
11.2.1.2	Source and Target Data Definitions	6
11.2.1.3	Required Data	6
11.2.1.4	Optional Data	8
11.2.2	Fixed Width Parsing	8
11.2.2.1	Header	9
11.2.2.2	Header and Record Data Type Translation	10
11.2.2.3	Key identification	11
11.2.2.4	Using a Source Definition File	11
11.2.3	Delimited Parsing	12
11.2.3.1	Metadata Columns	13
11.2.3.2	Parsing Properties	13
11.2.3.3	Parsing Steps	14
11.2.4	XML Parsing	14
11.2.4.1	Styles of XML	15
11.2.4.2	XML Parsing Rules	15
11.2.4.3	XPath Expressions	16
11.2.4.4	Other Value Expressions	18
11.2.4.5	Transaction Rules	18
11.2.4.6	Operation Rules	19
11.2.4.7	Column Rules	20
11.2.4.8	Overall Rules Example	21
11.2.5	Source Definitions Generation Utility	22
11.3	Message Capture Properties	22
11.3.1	Logging and Connection Properties	22
11.3.1.1	Logging Properties	22
11.3.1.2	JMS Connection Properties	23
11.3.1.3	JNDI Properties	25

11.3.2	Parser Properties	26
11.3.2.1	Setting the Type of Parser	26
11.3.2.2	Fixed Parser Properties	26
11.3.2.3	Delimited Parser Properties	31
11.3.2.4	XML Parser Properties	41

12 Troubleshoot

12.1	Troubleshooting the Java Adapters	1
12.1.1	Checking for Errors	1
12.1.2	Reporting Issues	2

Preface

This Article contains information about configuring, and running Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) to extend the capabilities of Oracle GoldenGate instances. Learn about Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) concepts and features, including how to setup and configure both the Classic as well as Microservices environments, and use the Handlers supported.

- [Audience](#)
- [Documentation Accessibility](#)
- [Conventions](#)
- [Related Information](#)

Audience

This guide is intended for system administrators who are configuring and running Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Accessible Access to Oracle Support

Oracle customers who have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Related Information

- [Oracle GoldenGate Product Documentation Libraries](#)
- [Oracle GoldenGate for Distributed Applications and Analytics](#)

1

Overview

- [Understanding Oracle GoldenGate for Distributed Applications and Analytics](#)
- [What's Supported in Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Dependency Downloader](#)
Utility scripts are located in the {GGforDAA install}/DependencyDownloader directory to download client dependency jars for the various supported Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) integrations.

1.1 Understanding Oracle GoldenGate for Distributed Applications and Analytics

This section describes the concepts and basic structure of the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

Watch this video for an introduction to Oracle GoldenGate Microservices: [Introduction to GoldenGate Microservices](#)

- [Understanding Oracle GoldenGate for Distributed Applications and Analytics](#)

1.1.1 Understanding Oracle GoldenGate for Distributed Applications and Analytics

The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) integrates with Oracle GoldenGate instances.

The Oracle GoldenGate product enables you to:

- Capture transactional changes from a source database.
- Sends and queues these changes as a set of database-independent files called the Oracle GoldenGate trail.
- Optionally alters the source data using mapping parameters and functions.
- Applies the transactions in the trail to a target system database.

Oracle GoldenGate performs this capture and apply in near real-time across heterogeneous databases, platforms, and operating systems.

- [Delivery Configuration Options](#)
- [Adapter Integration Options](#)
- [Monitoring Performance](#)

1.1.1.1 Delivery Configuration Options

The Java delivery module is loaded by the GoldenGate Replicat process, which is configured using the Replicat parameter file. Upon loading, the Java Delivery module is subsequently

configured based on the configuration present in the Adapter Properties file. Application behavior can be customized by:

- Editing the property files; for example to:
 - Set target types, host names, port numbers, output file names, JMS connection settings;
 - Turn on/off debug-level logging, and so on.
 - Identify which message format should be used.
- Records can be custom formatted by:
 - Setting properties for the pre-existing format process (for fixed-length or field-delimited message formats, XML, JSON, or Avro formats);
 - Customizing message templates, using the Velocity template macro language;
 - (Optional) Writing custom Java code.
- (Optional) Writing custom Java code to provide custom handling of transactions and operations, do filtering, or implementing custom message formats.

There are existing implementations (handlers) for sending messages using JMS and for writing out files to disk. For Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) targets, there are built in integration handlers to write to supported databases.

There are several predefined message formats for sending the messages (for example, XML or field-delimited); or custom formats can be implemented using templates. Each handler has documentation that describes its configuration properties; for example, a file name can be specified for a file writer, and a JMS queue name can be specified for the JMS handler. Some properties apply to more than one handler; for example, the same message format can be used for JMS and files.

1.1.1.2 Adapter Integration Options

There are two major products which are based on the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) architecture:

- The Oracle GoldenGate Java Adapter is the overall framework. This product allows you to implement custom code to handle Oracle GoldenGate trail records according to their specific requirements. It comes built-in with Oracle GoldenGate File Writer module that can be used for flat file integration purposes.
- GG for DAA. The GG for DAA product contains built-in support to write operation data from Oracle GoldenGate trail records into various GG for DAA targets (such as, HDFS, HBase, Kafka, Flume, JDBC, Cassandra, and MongoDB). You do not need to write custom code to integrate with GG for DAA applications. The functionality is separated into handlers that integrate with third party applications and formatters, which transform the data into various formats, such as Avro, JSON, delimited text, and XML. In certain instances, the integration to a third-party tool is proprietary, like the HBase API. In these instances, the formatter exists without an associated handler.

The Oracle GoldenGate Java Adapter and the GG for DAA products have some crossover in functionality so the handler exists without an associated formatter. The following list details the major areas of functionality and in which product or products the functionality is included:

- Read JMS messages and deliver them as an Oracle GoldenGate trail. This feature is included in GG for DAA.

- Read an Oracle GoldenGate trail and deliver transactions to a JMS provider or other messaging system or custom application. This feature is included in GG for DAA products.
- Read an Oracle GoldenGate trail and write transactions to a file that can be used by other applications. This feature is only included in GG for DAA.
- Read an Oracle GoldenGate trail and write transactions to a GG for DAA targets. The GG for DAA integration features are only included in GG for DAA product.
- [Capturing Transactions to a Trail](#)
- [Applying Transactions from a Trail](#)

1.1.1.2.1 Capturing Transactions to a Trail

Oracle GoldenGate message capture can be used to read messages from a queue and communicate with an Oracle GoldenGate Extract process to generate a trail containing the processed data.

The message capture processing is implemented as a Vendor Access Module (VAM) plug-in to a generic Extract process. A set of properties, rules and external files provide messaging connectivity information and define how messages are parsed and mapped to records in the target Oracle GoldenGate trail.

Currently this adapter supports capturing JMS text messages.

1.1.1.2.2 Applying Transactions from a Trail

Oracle GoldenGate Java Adapter delivery can be used to apply transactional changes to targets other than a relational database: for example, ETL tools (DataStage, Ab Initio, Informatica), JMS messaging, Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) Applications, or custom APIs. There are a variety of options for integration with Oracle GoldenGate:

- Flat file integration: predominantly for ETL, proprietary or legacy applications, Oracle GoldenGate File Writer can write micro batches to disk to be consumed by tools that expect batch file input. The data is formatted to the specifications of the target application such as delimiter separated values, length delimited values, or binary. Near real-time feeds to these systems are accomplished by decreasing the time window for batch file rollover to minutes or even seconds.
- Messaging: transactions or operations can be published as messages (for example, in XML) to JMS. The JMS provider is configurable to work with multiple JMS implementation; examples include ActiveMQ, JBoss Messaging, TIBCO, Oracle WebLogic JMS, WebSphere MQ, and others.
- Java API: custom handlers can be written in Java to process the transaction, operation and metadata changes captured by Oracle GoldenGate on the source system. These custom Java handlers can apply these changes to a third-party Java API exposed by the target system.
- GG for DAA integration: writing transaction data from the source trail files into various GG for DAA targets can be achieved by means of setting configuration properties. The GG for DAA product contains built in GG for DAA handlers to write to HDFS, HBase, Kafka, and Flume targets.

All four options have been implemented as extensions to the core Oracle GoldenGate product.

- For Java integration using either JMS or the Java API, use Oracle GoldenGate for Java.

- For GG for DAA integration, you can configure Oracle GoldenGate Replicat to integrate with the GG for DAA module. Writing to GG for DAA targets in various formats can be configured using a set of properties with no programming required.

1.1.1.3 Monitoring Performance

For more information about monitoring the performance, see [Monitor Performance from the Performance Metrics Service](#) in *Using Oracle GoldenGate Microservices Architecture*.

1.2 What's Supported in Oracle GoldenGate for Distributed Applications and Analytics

- [Verifying Certification and System Requirements](#)
Oracle recommends that you use the certification matrix and system requirements documents with each other to verify that your environment meets the requirements for installation.
- [Understanding Handler Compatibility](#)
- [What are the Additional Support Considerations?](#)

1.2.1 Verifying Certification and System Requirements

Oracle recommends that you use the certification matrix and system requirements documents with each other to verify that your environment meets the requirements for installation.

1. Verifying that your environment meets certification requirements:

Make sure that you install your product on a supported hardware and software configuration. See the certification matrix for more details: [GoldenGate Certifications](#).

Oracle has tested and verified the performance of your product on all certified systems and environments. Whenever new certifications are released, they are added to the certification document right away. New certifications can be released at any time. Therefore, the certification documents are kept outside the documentation libraries and are available on Oracle Technology Network.

2. Using the system requirements document to verify certification:

Oracle recommends that you use the [Oracle Fusion Middleware Supported System Configuration](#) document to verify that the certification requirements are met. For example, if the certification document indicates that your product is certified for installation on 64-Bit Oracle Linux 6.5, use this document to verify that your system meets the required minimum specifications. These include disk space, available memory, specific platform packages and patches, and other operating system-specific requirements. System requirements can change in the future. Therefore, the system requirement documents are kept outside of the documentation libraries and are available on Oracle Technology Network.

3. Verifying interoperability among multiple products:

To learn how to install and run multiple Fusion Middleware products from the same release or mixed releases with each other, see [Oracle Fusion Middleware Supported System Configuration](#) in *Oracle Fusion Middleware Understanding Interoperability and Compatibility*.

The compatibility of the GG for DAA handlers with the various data collections, including distributions, database releases, and drivers is included in the certification document.

1.2.2 Understanding Handler Compatibility

For more information, see the Certification Matrix.

1.2.3 What are the Additional Support Considerations?

This section describes additional Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) additional support considerations.

Pluggable Formatters—Support

The handlers support the Pluggable Formatters as follows:

- The File Writer Handler supports all of the pluggable formatters.
- The HDFS Handler supports all of the pluggable formatters.
- Pluggable formatters are not applicable to the HBase Handler. Data is streamed to HBase using the proprietary HBase client interface.
- The Kafka Handler supports all of the pluggable formatters.
- The Kafka Connect Handler does *not* support pluggable formatters. You can convert data to JSON or Avro using Kafka Connect data converters.
- The Kinesis Streams Handler supports all of the pluggable formatters described in the [Using the Pluggable Formatters](#).
- The Cassandra, MongoDB, and JDBC Handlers do *not* use a pluggable formatter.

Java Delivery Using Extract

Java Delivery using Extract is not supported. Support for Java Delivery is only supported using the Replicat process. Replicat provides better performance, better support for checkpointing, and better control of transaction grouping.

MongoDB Handler—Support

- The handler can only replicate unique rows from source table. If a source table has no primary key defined and has duplicate rows, replicating the duplicate rows to the MongoDB target results in a duplicate key error and the Replicat process abends.
- Missed updates and deletes are undetected so are ignored.
- Untested with sharded collections.
- Only supports date and time data types with millisecond precision. These values from a trail with microseconds or nanoseconds precision are truncated to millisecond precision.
- The datetime data type with timezone in the trail is not supported.
- A maximum BSON document size of 16 MB. If the trail record size exceeds this limit, the handler cannot replicate the record.
- No DDL propagation.
- No truncate operation.

JDBC Handler—Support

- The JDBC handler uses the generic JDBC API, which means any target database with a JDBC driver implementation should be able to use this handler. There are a myriad of

different databases that support the JDBC API and Oracle cannot certify the JDBC Handler for all targets.

- The handler supports Replicat using the `REPERROR` and `HANDLECOLLISIONS` parameters.
- DDL operations are ignored by default and are logged with a `WARN` level.
- Coordinated Replicat is a multithreaded process that applies transactions in parallel instead of serially. Each thread handles all of the filtering, mapping, conversion, SQL construction, and error handling for its assigned workload. A coordinator thread coordinates transactions across threads to account for dependencies. It ensures that DML is applied in a synchronized manner preventing certain DMLs from occurring on the same object at the same time due to row locking, block locking, or table locking issues based on database specific rules. If there are database locking issue, then Coordinated Replicat performance can be extremely slow or pauses.

DDL Event Handling

Only the `TRUNCATE TABLE` DDL statement is supported. All other DDL statements, such as `CREATE TABLE`, `CREATE INDEX`, and `DROP TABLE` are ignored.

You can use the `TRUNCATE` statements one of these ways:

- In a DDL statement, `TRUNCATE TABLE`, `ALTER TABLE TRUNCATE PARTITION`, and other DDL `TRUNCATE` statements. This uses the `DDL` parameter.
- Standalone `TRUNCATE` support, which just has `TRUNCATE TABLE`. This uses the `GETTRUNCATES` parameter.

1.3 Dependency Downloader

Utility scripts are located in the `{GGforDAA install}/DependencyDownloader` directory to download client dependency jars for the various supported Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) integrations.

Topics:

- [Dependency Downloader Setup](#)
- [Running the Dependency Downloader Scripts](#)
- [Dependency Downloader Scripts](#)

1.3.1 Dependency Downloader Setup

To complete the Dependency Downloader setup:

1. To verify that Java is installed, execute the following from the command line: `java -version`.

Note

The Dependency Downloader utility scripts require Java to run. Ensure that Oracle Java is downloaded and is available in the `PATH` on the machine where the scripts are installed.

2. Configure the proxy settings in the following script: `{GGforDAA install}/DependencyDownloader/config_proxy.sh`. Following are the 2 entries in this file:

- `#export PROXY_SERVER_HOST=www-proxy-hqdc.us.oracle.com`
- `#export PROXY_SERVER_PORT=80`

To configure the proxy settings:

- a. Uncomment the configuration settings. (remove the # at beginning of the lines).
- b. Change the host name and port number to your correct proxy server settings.

Note

Most companies maintain a private network which in turn has a network firewall to shield it from the public Internet. Additionally, most companies maintain a forwarding proxy server which serves as a gateway between the customer's private network and the public Internet. The Dependency Downloader utilities must access Maven repositories, which are available on the Internet. Therefore, you need to supply configuration for HTTP proxy settings in order to download dependency libraries. Proxy servers are identified by host name and port. If you do not know whether your company employs a proxy server or the settings, then contact your IT or network administrators.

The Dependency Downloader uses Bash scripts in order to invoke Maven and download dependencies. The Bash shell is not supported natively from the Windows Command Prompt. You can run the Dependency Downloader scripts on Windows, but it requires the installation of a Unix emulator. A Unix emulator provides a Unix style command line on Windows and supports various flavors of the Unix shells including Bash. An option for Unix emulators is Cygwin, which is available free of charge. After Cygwin is installed, the setup process is the same. Setup and running of the scripts should be done through the Cygwin64 Terminal. See <https://www.cygwin.com/>.

1.3.2 Running the Dependency Downloader Scripts

To run the dependency downloader scripts:

1. Use a Unix terminal interface navigate to the following directory: `{GGforDAA install}/DependencyDownloader`.
2. Execute the following to run the scripts: `./{the dependency script} {version of the dependencies to download}`

For example: `./aws.sh 1.11.893`

Dependency libraries get downloaded to the following directory:

`{GGforDAA install}/DependencyDownloader/dependencies/{the dependency name}_{the_dependency_version}`.

For example: `{GGforDAA install}/DependencyDownloader/dependencies/aws_sdk_1.11.893`.

Ensure that the version string exactly matches the version string of the dependency which is being downloaded. If a dependency version doesn't exist in the public Maven repository, then it is not possible to download the dependency and running the script results in an error. Most public Maven repositories support a web-based GUI whereby you can browse the supported versions of various dependencies. The exception is the Confluent Maven repository does not support a web-based GUI. This makes downloading dependencies challenging, because the version string is not independently verifiable through a web interface.

After the dependencies are successfully downloaded, you must configure the `gg.classpath` variable in the Java Adapter properties file to include the dependencies for the corresponding replicat process.

Note

Best Practices

1. Whenever possible, use the exact version of the client libraries to the server/application integration to which you are connecting.
2. Prior to running the Dependency Downloader scripts, independently verify that the version string exists in the repository through the web GUI.

1.3.3 Dependency Downloader Scripts

Table 1-1 Relevant Handlers/Capture

Client	Script	Description	Relevant Handlers/ Capture	Versions Supported	Dependen cy Link
Amazon AWS S3	s3.sh	This script downloads the client dependencies for AWS S3.	S3 Event Handler	2.x	https://mvnrepository.com/artifact/software.amazon.aws.sdk/s3
Amazon AWS Kinesis	kinesis.sh	This script downloads the client dependencies for AWS Kinesis.	Kinesis Handler	2.x	https://mvnrepository.com/artifact/software.amazon.aws.sdk/kinesis
Google BigQuery	bigquery.sh	This script downloads the required client libraries for Google BigQuery.	BigQuery Handler	2.x	https://search.maven.org/artifact/com.google.cloud/google-cloud-bigquery
Cassandra DSE (Datastax Enterprise) Client	cassandra_dse.sh	This script downloads the Cassandra DSE client. Cassandra DSE is the for-purchase version of Cassandra available from Datastax.	Cassandra Handler	2.0.0 and higher	https://search.maven.org/artifact/com.datastax.dse/dse-java-driver-core

Table 1-1 (Cont.) Relevant Handlers/Capture

Client	Script	Description	Relevant Handlers/ Capture	Versions Supported	Dependen cy Link
Apache Cassandra Client	cassandra.sh	This script downloads the Apache Cassandra client.	Cassandra Handler	4.0.0 and higher	https://search.maven.org/artifact/com.datastax.oss/java-driver-core
Cassandra Capture 3x Client	cassandra_capture_3x.sh	This script downloads all the client libraries needed for Capture from Cassandra 3.x versions.	Cassandra Capture 3x	3.3.1 (used by default)	https://mvnrepository.com/artifact/com.datastax.cassandra/cassandra-driver-core/3.3.1
Cassandra Capture 4x Client	cassandra_capture_4x.sh	This script downloads all the client libraries needed for Capture from Cassandra 4.x versions.	Cassandra Capture 4x	4.14.1 (used by default)	https://mvnrepository.com/artifact/com.datastax.oss/java-driver-core/4.14.1
Cassandra Capture DSE Client	cassandra_capture_dse.sh	This script downloads all the client libraries needed for Capture from DSE Cassandra 6.x versions.	Cassandra Capture DSE	4.14.1 (used by default)	https://mvnrepository.com/artifact/com.datastax.oss/java-driver-core/4.14.1
Elasticsearch Java Client	elasticsearch_java.sh	This script downloads the Elasticsearch Java Client.	Elasticsearch Handler	7.x and 8.x	https://search.maven.org/artifact/co.elastic.clients/elasticsearch-h-java

Table 1-1 (Cont.) Relevant Handlers/Capture

Client	Script	Description	Relevant Handlers/ Capture	Versions Supported	Dependen cy Link
Hadoop Azure Client from Cloudera	hadoop_azur e_cloudera. sh	This script downloads the Hadoop Azure client libraries provided by Cloudera. The Hadoop Azure client libraries cannot be loaded along with the Hadoop client because in Cloudera, the version numbers between the two components do not line up perfectly.	• HDFS Handler • HDFS Event Handler • ORC Event Handler • Parquet Event Handler	3.x	https:// repository.cl oudera.com /service/ rest/ repository/ browse/ cloudera- repos/org/ apache/ hadoop/ hadoop- azure/
Hadoop Client from Cloudera	hadoop_clo udera.sh	This script downloads the Hadoop client libraries provided by Cloudera.	• HDFS Handler • HDFS Event Handler • ORC Event Handler • Parquet Event Handler	3.x	https:// repository.cl oudera.com /service/ rest/ repository/ browse/ cloudera- repos/org/ apache/ hadoop/ hadoop- client/
Hadoop Client from Hortonworks	hadoop_hort onworks.sh	The Hadoop client including the libraries for connectivity to Azure Data Lake available from Hortonworks.	• HDFS Handler • HDFS Event Handler • ORC Event Handler • Parquet Event Handler	3.x	https:// repo.horton works.com/ service/ rest/ repository/ browse/ public/org/ apache/ hadoop/ hadoop- client/
Apache Hadoop Client Plus Required Libraries for Azure Connectivity	hadoop.sh	The Hadoop client including the libraries for connectivity to Azure Data Lake.	• HDFS Handler • HDFS Event Handler • ORC Event Handler • Parquet Event Handler	3.x	https:// search.mav en.org/ artifact/ org.apache. hadoop/ hadoop- azure

Table 1-1 (Cont.) Relevant Handlers/Capture

Client	Script	Description	Relevant Handlers/ Capture	Versions Supported	Dependen cy Link
HBase Client Provided by Cloudera	hbase_clo udera.sh	The HBase client libraries provided by Cloudera.	HBase Handler	2.x	https:// repository.cl oudera.com /service/ rest/ repository/ browse/ cloudera- repos/org/ apache/ hbase/ hbase- client/
HBase Client Provided by Hortonworks	hbase_horto nworks.sh	The HBase client libraries provided by Hortonworks.	HBase Handler	2.x	https:// repo.horton works.com/ service/ rest/ repository/ browse/ public/org/ apache/ hbase/ hbase- client/
Apache HBase Client	hbase.sh	The HBase client.	HBase Handler	2.x	https:// search.mav en.org/ artifact/ org.apache. hbase/ hbase- client
Apache Kafka Client plus Kafka Connect Framework and JSON Converter from Cloudera	kafka_clo udera.sh	The Kafka Client plus libraries for the Kafka Connect framework and the Kafka Connect JSON Converter provided by Cloudera.	• Kafka Handler • Kafka Connect Handler • Kafka Capture	0.9.x to current	https:// repository.cl oudera.com /service/ rest/ repository/ browse/ cloudera- repos/org/ apache/ kafka/ kafka- clients/

Table 1-1 (Cont.) Relevant Handlers/Capture

Client	Script	Description	Relevant Handlers/ Capture	Versions Supported	Dependen cy Link
Apache Kafka Client plus Kafka Connect Framework and JSON Converter from Hortonworks	kafka_hortonworks.sh	The Kafka Client plus libraries for the Kafka Connect framework and the Kafka Connect JSON Converter provided by Hortonworks.	• Kafka Handler • Kafka Connect Handler • Kafka Capture	0.9.x to current	https://repo.hortonworks.com/service/rest/repository/browse/public/org/apache/kafka/kafka-clients/
Apache Kafka Client plus Kafka Connect Framework and JSON Converter	kafka.sh	The Kafka Client plus libraries for the Kafka Connect framework and the Kafka Connect JSON Converter.	• Kafka Handler • Kafka Connect Handler • Kafka Capture	0.9.x to current	https://search.maven.org/artifact/org.apache.kafka/kafka-clients/
Confluent Kafka Client plus Kafka Connect Framework and JSON and Avro Converters	kafka_confluent.sh	The Kafka Client plus libraries for the Kafka Connect framework and the Kafka Connect JSON Converter and the Kafka Connect Avro Converter available from Confluent.	• Kafka Handler • Kafka Connect Handler • Kafka Capture	Confluent platform 4.1.0 and higher.	See https://packages.confluent.io/maven/io/confluent/kafka-connect-avro-converter/
MapR Kafka Client	kafka_mapr.sh	The MapR Kafka Client libraries.	Kafka Handler	0.x, 1.x, and 2.x	https://repository.mapr.com/nexus/content/groups/mapr-public/org/apache/kafka/kafka-clients/

Table 1-1 (Cont.) Relevant Handlers/Capture

Client	Script	Description	Relevant Handlers/ Capture	Versions Supported	Dependen cy Link
Confluent Kafka Client plus Kafka Connect Framework and Protobuf Converter	kafka_conf luent_protob uf.sh	The Kafka Client plus libraries for the Kafka Connect framework and the Kafka Connect Protobuf converter available from Confluent.	- Kafka Handler - Kafka Connect Handler	Confluent 5.x and higher	See https:// packages.c onfluent.io/ maven/io/ confluent/ kafka- connect- protobuf- converter/
MongoDB Client	mongodb.sh	The MongoDB client libraries.	MongoDB Handler	5.x	https:// mvnreposit ory.com/ artifact/ org.mongod b/mongodb- driver- legacy
Oracle NoSQL SDK Client	oracle_nosq l_sdk.sh	The Oracle NoSQL client libraries.	Oracle NoSQL Handler	5.x	https:// search.mav en.org/ artifact/ com.oracle. nosql.sdk/ nosqldriver
Oracle OCI Client	oracle_oci. sh	The Oracle OCI client libraries.	Oracle OCI Event Handler	3.x	https:// search.mav en.org/ artifact/ com.oracle. oci.sdk/oci- java-sdk- objectstora ge
Apache ORC (Optimized Row Columnar) Client	orc.sh	The Apache ORC client libraries. ORC is built on top of the Hadoop client so the ORC Event Handler needs the Hadoop client in order to run. The Hadoop client needs to be downloaded separately.	ORC Event Handler	1.x	https:// search.mav en.org/ artifact/ org.apache. orc/orc-core

Table 1-1 (Cont.) Relevant Handlers/Capture

Client	Script	Description	Relevant Handlers/ Capture	Versions Supported	Dependen cy Link
Apache Parquet Client	parquet.sh	The Apache Parquet client libraries. Parquet is built on top of the Hadoop client, so the Parquet Event Handler needs the Hadoop client in order to run. The Hadoop client needs to be downloaded separately.	Parquet Event Handler	1.x	https://search.maven.org/artifact/org.apache.parquet/parquet-hadoop
Apache Velocity	velocity.sh	The Velocity libraries were removed from the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) installation starting from the 21.1 release. This script downloads the libraries required for formatting using Velocity.	Velocity Formatter	1.x	https://search.maven.org/artifact/org.apache.velocity/velocity
Google Cloud Storage Java SDK	gcs.sh	This script downloads the required client libraries for Google Cloud Storage.	GCS Event Handler	2.x	https://search.maven.org/artifact/com.google.cloud/google-cloud-storage
MongoDB Capture	mongodb_capture.sh	This script downloads the required client libraries for MongoDB capture.	MongoDB Capture	5.x	https://search.maven.org/artifact/org.mongodb/mongodb-driver-reactivestreams

Table 1-1 (Cont.) Relevant Handlers/Capture

Client	Script	Description	Relevant Handlers/ Capture	Versions Supported	Dependen cy Link
Synapse JDBC Driver	synapse.sh	This script downloads the Synapse JDBC driver. Additionally, the Hadoop client is also required to stage data to Azure Data Lake.	Synapse Stage and Merge	12.6.1jre8	https://mvnrepository.com/artifact/com.microsoft.sqlserver/mssql-jdbc/12.6.1.jre8
Snowflake JDBC Driver	snowflake.sh	This script downloads the Snowflake JDBC driver. Other client libraries are likely required for staging the data to AWS or Azure cloud.	Snowflake Stage and Merge	3.15.1	https://search.maven.org/artifact/net.snowflake/snowflake-jdbc/3.15.1/jar
Jedis client for Redis	redis.sh	This script downloads Jedis which is a Redis client.	Redis Handler	4.x	https://search.maven.org/artifact/redis.clients/jedis
Google Pub/Sub Client	googlepubsub.sh	This script downloads the Java client for Google Pub/Sub Messaging.	Google Pub/Sub Handler	1.x	https://search.maven.org/artifact/com.google.cloud/google-cloud-pubsub
Databricks JDBC Driver	databricks.sh	This script downloads the Databricks JDBC driver.	Databricks Stage and Merge	2.6.36	https://mvnrepository.com/artifact/com.databricks/databricks-jdbc/2.6.36
Azure Blob Storage Client	azure_blob_storage.sh	This script downloads the Microsoft Azure Blob Storage Client.	Azure Blob Storage Event Handler Data Warehouse Stage and Merge implementations can use this as well to upload to Azure Data Lake.	12.25.3	https://search.maven.org/artifact/com.azure/azure-storage-blob

Table 1-1 (Cont.) Relevant Handlers/Capture

Client	Script	Description	Relevant Handlers/ Capture	Versions Supported	Dependen cy Link
Snowflake Streaming	snowflakestreaming.sh	This script can be downloaded using the Dependency Downloader script.	NA	NA	The script can be found in following location :<OGGDIR>/DependencyDownloader/snowflakestreaming.sh

2

Prepare

- [Preparing for Installation](#)

2.1 Preparing for Installation

Prepare your Java environment by ensuring that you have the correct version of Java installed, and that the environmental variables have been set up and configured correctly.

- [Downloading Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Installation Overview](#)
- [Directories and Variables in Microservices Architecture](#)
- [Setting up Environmental Variables](#)

2.1.1 Downloading Oracle GoldenGate for Distributed Applications and Analytics

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) are available for Windows, Linux, and UNIX. To download, first visit the Oracle support site to see if there is a patch available for your operating system and architecture. See also, [GoldenGate Certification Matrix](#).

Note

If you are not planning to use the generic build included in the installation, ensure that the major release of the GG for DAA build you download matches (or is known to be compatible with) the major release of the Oracle GoldenGate instance that will be used with it.

1. Navigate to <http://support.oracle.com>.
2. Sign in with your Oracle ID and password.
3. Select the **Patches and Upgrades** tab.
4. On the **Search** tab, click **Product or Family**.
5. In the **Product** field, type **Oracle GoldenGate for Distributed Applications and Analytics**.
6. From the **Release** drop-down list, select the release version that you want to download.
7. Make sure Platform is displayed as the default in the next field, and then select the platform from the drop-down list.
8. Leave the last field blank.
9. Click **Search**.

10. In the **Advanced Patch Search Results** list, select the available builds that satisfy the criteria that you supplied.
11. In the file **Download** dialog box, click the ZIP file to begin the download.

If patches are not available on the support site, go to the Oracle delivery site for the release download.

1. Navigate to <http://edelivery.oracle.com>.
2. Sign in with your Oracle ID and password.
3. On the Terms and Restrictions page:
 - Accept the **Trial License Agreement** (even if you have a permanent license).
 - Accept the **Export Restrictions**.
 - Click **Continue**.
4. On the **Media Pack Search** page:
 - Select the Oracle Fusion Middleware Product Pack.
 - Select the platform on which you will be installing the software.
 - Click **Go**.
5. In the **Results** list:
 - Select the **Oracle GoldenGate for Distributed Applications and Analytics**.
 - Click **Continue**.
6. On the **Download** page:
 - View the Readme file.
 - Click **Download** for each component that you want. Follow the automatic download process to transfer the zip file to your system.

2.1.2 Installation Overview

This section provides an overview of the installation contents and the Oracle GoldenGate instances used with the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

- [Contents of the Installation ZIP File](#)
- [Using the Generic Build of Oracle GoldenGate](#)
- [Considerations for Using a Custom Build for a GoldenGate for Distributed Applications and Analytics Instance of Oracle GoldenGate](#)
- [Installing to a Non-Generic Instance of Oracle GoldenGate](#)

2.1.2.1 Contents of the Installation ZIP File

The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) installation ZIP file contains:

- Oracle GoldenGate Java Adapter
- A version of Oracle GoldenGate designed to stream data to supported targets. This version is labeled *generic* because it is not specific to any database, but it is platform dependent. For more information, see [GoldenGate Certification Matrix](#).

2.1.2.2 Using the Generic Build of Oracle GoldenGate

For JMS capture, the Java Adapter must run in the generic build of Oracle GoldenGate. However, the generic build is not required when using the adapter for delivery of trail data to a target; in this case, the Java Adapter can be used with any database version of Oracle GoldenGate.

2.1.2.3 Considerations for Using a Custom Build for a GoldenGate for Distributed Applications and Analytics Instance of Oracle GoldenGate

There are both advantages and disadvantages to installing a custom build for an Oracle GoldenGate instance. Also, there are limitations in the releases of Oracle GoldenGate that are compatible with releases of the GG for DAA.

Advantages

- The non-generic instance allows you to configure Extract to login to the database for metadata. This removes the need to use a source definitions file that must be synchronized with the source database DDL.
- There is no need to manage two separate versions of Oracle GoldenGate when doing database capture and JMS delivery on the same server.

Disadvantages

- If you need to patch Oracle GoldenGate core instance, you must also copy GG for DAA into the new patched installation of Oracle GoldenGate.
- The GG for DAA is only tested and certified with the generic version of Oracle GoldenGate core. New patches of the core can trigger incompatibilities.

Limitations

- The Replicat module to write to GG for DAA targets is only available in the Generic Oracle GoldenGate distribution.
- The generic build must be used with JMS capture, as this is the only version of Extract that is capable of loading the VAM.
- A DEFGEN utility is not included with GG for DAA. To generate source definitions, you will need a version of Oracle GoldenGate that is built specifically for your database type.

2.1.2.4 Installing to a Non-Generic Instance of Oracle GoldenGate

If you decide to install the Java user exit to a non-generic instance of Oracle GoldenGate, unzip to a temporary location first and then copy the adapter files to your Oracle GoldenGate installation location.

To install the Java user exit to a non-generic instance of Oracle GoldenGate:

1. Navigate to the Oracle GoldenGate installation directory, for example C:\ggs.
2. Create a temporary directory, and extract the Java user exit ZIP file into this sub directory within it, for example ggjava.
3. Copy or move the files from the ggjava sub directory and shared libraries into the Oracle GoldenGate installation directory (C:\ggs).

Note

You need not copy the shared library `ggjava_vam` because, it only works with the generic build.

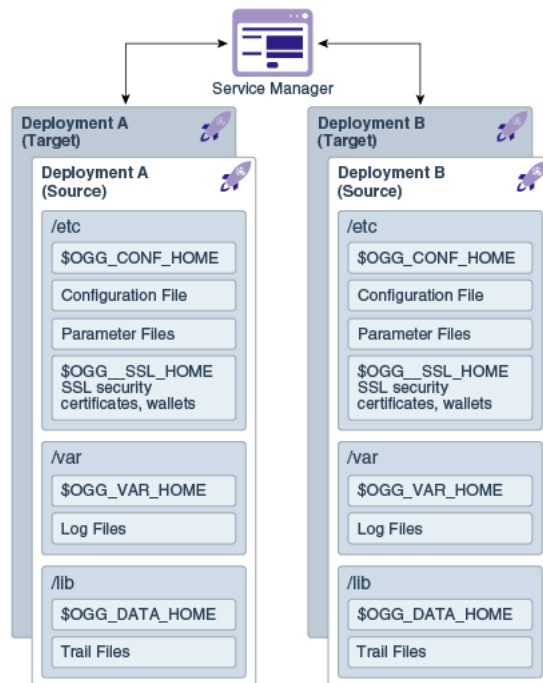
2.1.3 Directories and Variables in Microservices Architecture

The Microservices Architecture is designed with a simplified installation and deployment directory structure.

This directory structure is based on the Linux Foundation Filesystem Hierarchy Standard. Additional flexibility has been added to allow parts of the deployment subdirectories to be placed at other locations in the file system or on other devices, including shared network devices. The design comprises a read-only Oracle GoldenGate home directory where Oracle GoldenGate Microservices Architecture is installed and custom deployment specific directories are created as follows:

- `bin`
- `cfgtoollogs`
- `deinstall`
- `diagnostics`
- `include`
- `install`
- `inventory`
- `jdk`
- `jlib`
- `lib`
 - `instantclient`
 - `sql`
 - `utl`
- `OPatch`
- `oraInst.loc`
- `oui`
- `srvm`

The following figure shows the files and directories under the Services Manager (`srvm`) directory:

Figure 2-1 GoldenGate MA Directory Structure

The following table describes the key MA directories and the variables that are used when referring to those directories during an Oracle GoldenGate installation. When you see these variables in an example or procedure, replace the variable with the full path to the corresponding directory path in your enterprise topology.

Directory Name	Variable	Description	Default Directory Path
Oracle GoldenGate home	OGG_HOME	The Oracle GoldenGate home that is created on a host computer is the directory that you choose to install the product, here GoldenGate for Distributed Applications and Analytics (GG for DAA). This read-only directory contains binary, executable, and library files for GG for DAA.	/ogg_install_location
Deployment etc home	OGG_ETC_HOME	The location where your deployment configuration files are stored including parameter files.	/ogg_deployment_location/etc
Deployment configuration home	OGG_CONF_HOME	The location where each deployment information and configuration artifacts are stored.	/ogg_deployment_location/etc/conf

Directory Name	Variable	Description	Default Directory Path
Deployment security home	OGG_SSL_HOME	The location where each deployment security artifacts (certificates, wallets) are stored.	/ogg_deployment_location/etc/ssl
Deployment variable home	OGG_VAR_HOME	The location where each deployment logging and reporting processing artifacts are stored.	/ogg_deployment_location/var
Deployment data home	OGG_DATA_HOME	The location where each deployment data artifacts (trail files) are stored.	/ogg_deployment_location/var/lib/data

You can change the default location of all of these to customize where you want to store these files.

In a configuration where the OGG_VAR_HOME is a local directory and the OGG_HOME is a shared read-only remote directory, many deployments with local OGG_VAR_HOME can share one read-only shared OGG_HOME.

This directory design facilitates a simple manual upgrade. To upgrade, you stop the services and then set the OGG_HOME in the web interface (or via a REST command) and then restart the processes. On restart, Oracle GoldenGate picks up the updated environment variables. You simply switch a deployment to use a new Oracle GoldenGate release by changing the Oracle GoldenGate home directory path in the Service Manager to a new Oracle GoldenGate home directory, which completes the upgrade. Restart the microservices, Extract and Replicat processes.

2.1.4 Setting up Environmental Variables

To configure your Java environment for Oracle GoldenGate for Java:

- The PATH environmental variable should be configured to find your Java Runtime
- The shared (dynamically linked) Java virtual machine (JVM) library must also be found.

On Windows, these environmental variables should be set as system variables; on Linux/UNIX, they should be set globally or for the user running the Oracle GoldenGate processes. Examples of setting these environmental variables for Windows, UNIX, and Linux are in the following sections.

Note

There may be two versions of the JAVA_HOME/.../client, and another in JAVA_HOME/.../server. For improved performance, use the server version, if it is available. On Windows, only the client JVM may be there if only the JRE was installed (and not the JDK).

- [Java on Linux/UNIX](#)
- [Java on Windows](#)

2.1.4.1 Java on Linux/UNIX

Configure the environment to find the JRE in the PATH, and the JVM shared library, using the appropriate environmental variable for your system. For example, on Linux (and Solaris), set LD_LIBRARY_PATH to include the directory containing the JVM shared library as follows (for sh/ksh/bash):

Note

On AIX platforms, you set LIBPATH=. On HP-UX IA64, you set SHLIB_PATH=.

Example 2-1 Configuring path for Java on Linux

```
export JAVA_HOME=/opt/jdk1.8
export PATH=$JAVA_HOME/bin:$PATH
export LD_LIBRARY_PATH=$JAVA_HOME/jre/lib/amd64/server:$LD_LIBRARY_PATH
```

In this example, the directory \$JAVA_HOME/jre/lib/i386/server should contain the libjvm.so and libjsig.so files. The actual directory containing the JVM library depends on the operating system and if the 64-bit JVM is being used.

Verify the environment settings by opening a command prompt and checking the Java version as in this example:

```
$ java -version
java version "1.8.0_92"
Java(TM) SE Runtime Environment (build 1.8.0_92-b14)
```

2.1.4.2 Java on Windows

After Java is installed, configure the PATH to find the JRE and JVM DLL (jvm.dll):

Example 2-2 Configuring Path for Java on Windows

```
set JAVA_HOME=C:\Program Files\Java\jdk1.8.0
set PATH=%JAVA_HOME%\bin;%PATH%
set PATH=%JAVA_HOME%\jre\bin\server;%PATH%
```

In the example above, the directory %JAVA_HOME%\jre\bin\server should contain the file jvm.dll.

Verify the environment settings by opening a command prompt and checking the Java version as in this example:

```
C:\> java -version
java version "1.8.0_92" Java(TM) SE Runtime Environment (build 1.8.0_92-b14))
```

3

Install

- [Setting up Oracle GoldenGate for Distributed Applications and Analytics in a High Availability Environment](#)
- [Installing Oracle GoldenGate for Distributed Applications and Analytics](#)
The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) is installed using OUI. You can also use a command line silent installation using OUI.

3.1 Setting up Oracle GoldenGate for Distributed Applications and Analytics in a High Availability Environment

This topic describes the best practices of achieving high availability of Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) processes.

Topics:

- [Running GG for DAA from a Single Instance](#)
- [Running GG for DAA on a Cluster of Servers](#)
- [Shared Storage](#)
Most shared storage solutions, including general purpose cluster file systems, can be used to install Oracle GoldenGate or to store the files that Oracle GoldenGate needs to recover.

3.1.1 Running GG for DAA from a Single Instance

To configure the single server high availability, you need to configure the manager process with AUTOSTART and AUTORESTART parameters. These parameters ensure that the manager process always gets the extract or replicat group to be started back up from an inactive state.

3.1.2 Running GG for DAA on a Cluster of Servers

Depending on which cluster manager software that is being used, you need to configure it to ensure the following:

- There is **exactly one active node** that is running Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA). It is assumed that the cluster manager can detect that a compute node is down and subsequently spawn another node to be the active.
- Install GG for DAA in shared file system and have that shared file system mounted in the same location for all the nodes participating in the High Availability (HA) configuration. For more information about installing Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA), see [Installing Oracle GoldenGate for Distributed Applications and Analytics](#). Most of the state files, including the Input and Output Trail files, Configuration files, and Checkpoint files described in the next point are stored in sub-directories of the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) install. The GG for DAA installation directory is the same across all managed nodes. This helps the administrator to leverage the exact content of entry point script to bring up GG for DAA as

part of its workflow to spawn a new active node. An example of the content of the entry point script is a command to start the Oracle GoldenGate manager process.

- Oracle GoldenGate artifacts are stored in one or more **shared file systems or volumes accessible from all nodes**. For more information about these files, see [Directories and Variables in Microservices Architecture](#):
 - **Input and Output Trail files**: Typically these files are located in the `gg_install_dir/dirdat` directory, where `gg_install_dir` is the Oracle GoldenGate installation directory, such as `C:/ggs` on Windows or `/home/user/ggs` on UNIX. These files are configurable.
 - **Configuration files**: The configuration files are located in the `gg_install_dir/dirprm` directory.
 - **Checkpoint files**: These files are stored in an internal subdirectory, such as the `gg_install_dir/dirchk` directory.
 - When using File Writer features, for example, File Writer handler, ADW, or Redshift integration, the file writer output files and the state files must be on shared volumes.

For more information about configuring cluster high availability for handlers, see [Configuring Cluster High Availability](#).

3.1.3 Shared Storage

Most shared storage solutions, including general purpose cluster file systems, can be used to install Oracle GoldenGate or to store the files that Oracle GoldenGate needs to recover.

The following options are available from Oracle:

- **Oracle Cluster File System (OCFS2) –available only on Linux**: OCFS2 can also be used for Oracle Database storage, although Oracle recommends the use of Oracle Automatic Storage Management (ASM) starting with Oracle Database 10g. For more information, see <http://oss.oracle.com/projects/ocfs2/>.
- **Oracle Automatic Storage Management (ASM) Cluster File System (ACFS)**: For more information about the Oracle Database 11g Release 2 ACFS, see Oracle Database Automatic Storage Administrator's Guide as part of the Oracle Database 11g Release 2 documentation set: https://docs.oracle.com/cd/E11882_01/server.112/e18951/asmfs_util001.htm#OSTMG91000.
- **Oracle Database File System (DBFS)**: For more information about DBFS, its restrictions as well as how to configure a DBFS, see Oracle Database Secure File and Large Objects Developer's Guide from the Oracle Database 11g Release 2 documentation set: https://docs.oracle.com/cd/E11882_01/appdev.112/e18294/adlob_fs.htm#BABDHGGJ.
- **Oracle ACFS with Oracle Database 11g Release 2**

3.2 Installing Oracle GoldenGate for Distributed Applications and Analytics

The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) is installed using OUI. You can also use a command line silent installation using OUI.

This chapter describes how to install a new instance of GG for DAA. The Installation is a three-step process:

- Install the GG for DAA Microservices Architecture (MA).

- Set the necessary environment variables.
- Deploy an Oracle GoldenGate instance using the configuration assistant.

The installer registers the Oracle GoldenGate home directory with the central inventory that is associated with the selected database. The inventory stores information about all Oracle software products installed on a host if the product was installed using OUI.

Disk space is also required for the Oracle GoldenGate Bounded Recovery feature. Bounded Recovery is a component of the general Extract checkpointing facility. It caches long-running open transactions to disk at specific intervals to enable fast recovery upon a restart of Extract. At each bounded recovery interval (controlled by the BRINTERVAL option of the BR parameter) the disk required is as follows: for each transaction with cached data, the disk space required is usually 64k plus the size of the cached data rounded up to 64k. Not every long-running transaction is persisted to disk.

Watch this video for a demo on installing and configuring [Install and Configure GoldenGate Microservices 21c](#).

Topics:

- [Installing Oracle GoldenGate MA for Distributed Applications and Analytics Using the UI](#)
Interactive installation provides a graphical user interface that prompts for the required installation information. These instructions apply to new installations and upgrades.
- [Silent Installation](#)
Silent installation from the command line interface can be performed if your system does not have an X-Windows or graphical interface or you want to perform the installation in an automated way. Silent installations ensure that multiple users in your organization use the same installation options when installing Oracle products.
- [Setting Up Secure or Non-Secure Deployments](#)
You can choose to set up a secure or non-secure deployment.

3.2.1 Installing Oracle GoldenGate MA for Distributed Applications and Analytics Using the UI

Interactive installation provides a graphical user interface that prompts for the required installation information. These instructions apply to new installations and upgrades.

To install Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) using the UI:

1. Create a temporary staging directory into which you will install Oracle GoldenGate. For example, `mkdir /u01/stage/oggsc`.
2. Extract the installation ZIP file into the temporary staging directory. For example: `unzip ggs_Linux_x64_BigData_64bit_services_shiphome.zip -d ./temp directory`
3. From the expanded directory, run the `ggs_Linux_x64_BigData_64bit_services/Disk1/runInstaller` program on UNIX or Linux to display the **Installation Wizard**.
4. On the **Select Installation Option** page, select the Oracle Database version for your environment, then click **Next**.
5. If you are on Windows and running Manager as a service, set the system variable PATH to include `jvm.dll`, then delete the Manager service and re-add it.
6. On the **Specify Installation Details** page, ensure that the following environment variable is set:
 - OGG_HOME

7. Click **Next** to display the **Summary** page.
 8. Confirm that there is enough space for the installation and that the installation selections are correct.
 - (Optional) Click **Save Response File** to save the installation information to a response file. You can run the installer from the command line with this file as input to duplicate the results of a successful installation on other systems. You can edit this file or create a new one from a template.
 - Click **Install** to begin the installation or **Back** to go back and change any input specifications. When upgrading an existing Oracle GoldenGate installation, OUI notifies you that the software location has files or directories. Click **Yes** to continue.
 - If you created a central inventory directory, then you are prompted to run the `INVENTORY_LOCATION/orainstRoot.sh` script. This script must be executed as the root operating system user. This script establishes the inventory data and creates subdirectories for each installed Oracle product (in this case, Oracle GoldenGate).

You are notified when the installation is completed.
 9. Click **Close** to complete the installation.
- Watch this video on installing [GoldenGate Microservices](#).

3.2.2 Silent Installation

Silent installation from the command line interface can be performed if your system does not have an X-Windows or graphical interface or you want to perform the installation in an automated way. Silent installations ensure that multiple users in your organization use the same installation options when installing Oracle products.

Silent installations are driven by using a response file. Response files can be saved by selecting the Save Response File option during an interactive Oracle Universal Installer session or by editing the `oggcore.rsp` template located in the response directory after unzipping the binaries.

The Oracle GoldenGate response file contains a standard set of Oracle configuration parameters in addition to parameters that are specific to Oracle GoldenGate. These parameters correspond to the fields in the interactive session. The response file location is

`unzipped_directory/ggs_Linux_x64_BigData_64bit_services/Disk1/response`

To perform the installation using a response file, issue the following command:

```
unzipped_directory/ggs_Linux_x64_BigData_64bit_services.zip/Disk1/runInstaller -silent -  
nowait -responseFile absolute_path_to_response_file
```

3.2.3 Setting Up Secure or Non-Secure Deployments

You can choose to set up a secure or non-secure deployment.

A secure deployment involves making RESTful API calls and conveying trail data between the Distribution Server and Receiver Server, over SSL/TLS. You can use your own existing business certificate from your Certificate Authority (CA) or you might create your own certificates. When first creating the SSL/TLS security certificates, you must ensure that the SSL/TLS security environment variables.

For a non-secure deployment, the RESTful API calls occur over plain-text HTTP and conveyance between Distribution Server and Receiver Server is performed using the wss, ogg, and ws protocols.

This section describes the steps to configure a non-secure deployment and prerequisites and tasks to configure a secure deployment.

- [How to Add Secure or Non-Secure Deployments](#)
Adding deployments is the first task in the process of setting up a data replication platform. Deployments are managed from the Service Manager.
- [How to Remove a Deployment](#)
You can remove a deployment using OGGCA or in silent mode.

3.2.3.1 How to Add Secure or Non-Secure Deployments

Adding deployments is the first task in the process of setting up a data replication platform. Deployments are managed from the Service Manager.

After completing the Oracle GoldenGate MA installation, you can add initial and subsequent deployments using the Configuration Assistant (OGGCA) wizard.

Note

Oracle recommends that you have a single Service Manager per host, to avoid redundant upgrade and maintenance tasks with Oracle GoldenGate releases.

Use OGGCA to add multiple deployments to a Service Manager. This allows you to upgrade the same Service Manager with new releases or patches. The source and target deployments serve as endpoints for setting up the distribution path for data replication.

1. From the OGG_HOME directory, run the \$OGG_HOME/bin/oggca.sh program on UNIX or Linux.
The Oracle GoldenGate Configuration Assistant (oggca) is started. Run this program, each time you want to add a deployment.
2. In the **Select Service Manager Options** step:
 - a. Select whether you want to use an existing Service Manager or create a new one. In most configurations, you only have one Service Manager that is responsible for multiple deployments.
 - b. For a new Service Manager, enter or browse to the directory that you want to use for your deployment. Oracle recommends that you create a ServiceManager directory within the deployment sub-directory structure to store the Service Manager files.
 - c. Enter the hostname or IP Address of the server.
 - d. Enter a unique port number that the Service Manager will listen on, or choose the port already in use if selecting an existing Service Manager.
 - e. (Optional) You can register the Service Manager to run as a service so as to avoid manually starting and stopping it.

You can choose to run *one* Service Manager as a service (daemon). If there is an existing Service Manager registered as a service and you select a new Service Manager to register as a service, an alert is displayed indicating that you cannot register the new one as a service. All other Service Managers are started and stopped using scripts installed in the bin directory of the deployment. You cannot register an existing Service Manager as a service.

3. In the **Configuration Options** step, you can add or remove deployments.
You can only add or remove one deployment for one Service Manager at a time.

Note

Ensure that your Service Manager is up and running prior to launching OGGCA.

4. In the **Deployment Details** step:
 - a. Enter the deployment name using these conventions:
 - Must begin with a letter.
 - Can be a standard ASCII alphanumeric string not exceeding 32 characters.
 - Cannot include extended ASCII characters.
 - Special characters that are allowed include underscore ('_'), hyphen ('/'), dash ('-'), period ('.'). The name before the / symbol should be "slash" or "forward slash".
 - Cannot be "ServiceManager".
 - b. Enter or select the Oracle GoldenGate installation directory. If you have set the \$OGG_HOME environment variable, the directory is automatically populated. Otherwise, the parent directory of the oggca.sh (Linux) or oggca.bat (Windows) script is used.
 - c. Click **Next**.
5. On the **Select Deployment Directories** page:
 - a. Enter or select a deployment directory where you want to store the deployment registry and configuration files. When you enter the deployment directory name, it is created if it doesn't exist. Oracle recommends that you do *not* locate your deployment directory inside your \$OGG_HOME and that you create a separate directory for easier upgrades. The additional fields are automatically populated based on the specified deployment directory.

Note

The deployment directory name (user deployment directory) needs to be different than the directory name chosen in the first screen (Service Manager deployment directory).

- b. You can customize the deployment directories so that they are named and located differently from the default.
 - c. Enter or select different directories for the various deployment elements.
 - d. Click **Next**.
6. On the **Environment Variables** page:

Enter the requested values for the environment variables. Double-click in the field to edit it. You can copy and paste values in the environment variable fields. Make sure that you tab or click outside of the field after entering each value, otherwise it's not saved. If you have set any of these environment variables, the directory is automatically populated.

OGG_HOME

The directory where you installed Oracle GoldenGate. This variable is fixed and cannot be changed.

Note

On a Windows platform, ensure that there's no space in the OGG_HOME directory path otherwise OGGCA will not run.

LD_LIBRARY_PATH

This variable is used to specify the path to search for libraries on UNIX and Linux. It may have a different name on some operating systems, such as LIBPATH on IBM AIX on POWER Systems (64-Bit), and SHLIB_PATH on HP-UX. This path points to the Oracle GoldenGate installation directory and the underlying instant client directory by default. It might be extended if USER_EXITS are in use.

You can add additional environment variables to customize your deployment or remove variables. For instance, you can enter the following variable to default to another international charset: `ENV_LC_ALL=zh_CN.UTF-8`

Click **Next**.

7. On the Administrator Account page:

- a. Enter a user name and password that you want to use to sign in to the Oracle GoldenGate MA Service Manager and the other servers. This user is the security user for this deployment. Select the **Enable strong password policy in the new deployment** checkbox to ensure setting a highly secure password for your user account. The strong password policy has the following requirements:
 - At least one lowercase character [a...z]
 - At least one uppercase character [A...Z]
 - At least one digit [0...9]
 - At least one special character [- ! @ % & * . #]
 - The length should be between 8 and 30 characters.

If you are using an existing Service Manager, you must enter the same log in credentials that were used when adding the first deployment.

- b. Select the check box that allows you to enable a strong password policy for your new deployment. If you select this option, then the password must adhere to restrictions, otherwise an error occurs, which requires you to specify a stronger password.
- c. Click **Next**.

8. On the Security Options page:

- a. You can choose whether or not you want to secure your deployment. Oracle recommends that you enable SSL/TLS security. If you do not want to use security for your deployment, deselect the check box.

This operation exposes the option **This non-secure deployment will be used to send trail data to a secure deployment**. Select this check box if the non-secure target deployment is meant to communicate with a secure source deployment.

However, you must enable security if configuring for Oracle GoldenGate sharding support.

- b. Also see: [About Target-Initiated Paths](#) in *Step by Step Data Replication Using Oracle GoldenGate Microservices Architecture Guide*.
 - c. (Optional) You can specify a client wallet location so that you can send trail data to a secure deployment. This option is useful when Distribution Server from the source deployment is unsecured whereas the Receiver Server on the target deployment is secured. So, the sender may be configured for public access while the Receiver Server requires authentication and authorization, which is established using PKI before the incoming data is applied. For more information, see [Creating a Self Signed Certificate](#) and [Creating a Client Certificate](#) Certificate in *Oracle GoldenGate Security Guide*.
 - d. For your Server, select one of the options, and then provide the required file locations. When using an existing wallet, it must have the appropriate certificates already imported into it. If you choose to use a certificate, enter the corresponding pass phrase.
When using a self-signed certificate, a new Oracle Wallet is created in the new deployment and these certificates are imported into it. For certificates, enter the location of the private key file and the pass phrase. The private key files must be in the PKCS#8 format.
 - e. For your Client, select one of the options, and then provide the required information as you did for your server.
 - f. Click **Next**.
9. (If Security is enabled) On the **Advanced Security Settings** page, the TLS 1.1 and TLS 1.2 options are available. TLS 1.2 is selected by default.

When you open the Advanced Security Settings for the first time with TLS 1.2, the following cipher suites are listed:

```
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
```

- a. Use the arrows to add or remove cipher suites.

- b. Use **Up** and **Down** to reorder how the cipher suites are applied
 - c. Click **Next**.
10. (If Sharding is enabled) On the **Sharding Options** page:
- a. Locate and import your Oracle GoldenGate Sharding Certificate. Enter the distinguished name from the certificate that will be used by the database sharding code to identify itself when making REST API calls to the Oracle GoldenGate MA services.
 - b. Enter a unique name for the certificate.
 - c. Click **Next**.
11. On the **Port Settings** page:
- a. Enter the Administration Server port number, and then when you leave the field the other port numbers are populated in ascending numbers. Optionally, you can enter unique ports for each of the servers.
 - b. Select **Enable Monitoring** to use the Performance Metrics Server.
 - c. Click inside the Performance Metrics Server port fields to populate or enter the ports you want to use. Ensure that you choose available ports for TCP.

Select the UDP port for performance monitoring. The option to select the UDP port is displayed only with deployments on Windows and other operating systems that don't support UDS communication with Performance Metric Server.

You can change the TCP port from the Service Manager console after the deployment is done. For more information on PMSRVR, see ENABLEMONITORING.
 - d. Select the type of datastore that you want the Performance Metrics Server to use, the default Berkeley Database (BDB) data store or Open LDAP Lightning Memory-Mapped Database (LMDB). You can also designate the Performance Monitor as a Critical Service if integrating the Service Manager with XAG.

For LMDB information, see <http://www.lmdb.tech/doc/>.
 - e. Select the location of your datastore. BDB and LMDB are in-memory and disk-resident databases. The Performance Metrics server uses the datastore to store all performance metrics information.
 - f. Click **Next**.

Note

The oggca utility validates whether or not the port you entered is currently in use or not.

12. On the **Summary** page:
- a. Review the detailed configuration settings of the deployment before you continue.
 - b. (Optional) You can save the configuration information to a response file. Oracle recommends that you save the response file. You can run the installer from the command line using this file as an input to duplicate the results of a successful configuration on other systems. You can edit this file or a new one from the provided template.

Note

When saving to a response file, the administrator password is not saved for security reasons. You must edit the response file and enter the password if you want to reuse the response file for use on other systems.

- c. Click **Finish** to the deployment.
 - d. Click **Next**.
13. On the **Configure Deployment** page:
- Displays the progress of the deployment creation and configuration.
- a. If the Service Manager is being registered as a service, a pop-up appears that directs you how to run the script to register the service. The Configuration Assistant verifies that these scripts have been run. If you did not run them, you are queried if you want to continue. When you click **Yes**, the configuration completes successfully. When you click **No**, a temporary failed status is set and you click **Retry** to run the scripts.
- Click **Ok** after you run the script to continue.
- b. Click **Next**.
14. On the **Finish** page:
- Click **Close** to close the Configuration Assistant.

3.2.3.2 How to Remove a Deployment

You can remove a deployment using OGGCA or in silent mode.

Topics:

- [How to Remove a Deployment: GUI](#)
You can remove a deployment using the Oracle GoldenGate Configuration Assistant wizard.
- [How to Remove a Deployment: Silent Mode](#)
You can remove a deployment silently using the Oracle GoldenGate Configuration Assistant (oggca) from the Oracle GoldenGate Home bin directory.

3.2.3.2.1 How to Remove a Deployment: GUI

You can remove a deployment using the Oracle GoldenGate Configuration Assistant wizard.

To remove a deployment:**Note**

When you remove a deployment or uninstall Oracle GoldenGate MA, the system does not automatically stop processes. As a result, you may have to stop processes associated with the deployment and you must clean files manually.

1. Run the Oracle GoldenGate Configuration Assistant wizard:
`$OGG_HOME/bin`

2. Select **Existing Service Manager** from the **Select Service Manager Options** screen. Click **Next**.
3. Select **Remove Existing Oracle GoldenGate Deployment** from the Configuration Options screen.
4. Select the deployment you need to remove from the **Deployment Name** list box. Also select the **Delete Deployment Files from Disk** check box if you want to remove all the deployment files (including configuration files) from the host.
5. Enter the Administration account user name and password and click **Next**.
6. See the list of settings that are deleted with the deployment and click **Finish**.

To remove a Service Manager:

1. Run Oracle GoldenGate Configuration Assistant wizard:
\$OGG_HOME/bin
2. Select **Existing Service Manager** from the **Select Service Manager Options** screen. Click **Next**.
3. If there are no other deployments to remove, then the option to remove the Service Manager is available in the drop down. Select **Remove Service Manager Deployment** from the Configuration Options screen.
4. Click **Finish**.

Files to be Removed Manually After Removing Deployment

It's mandatory to delete some files manually only in case there's a Service Manager registered but you have to unregister it and register a new one. To remove files manually, you must have root or sudo privileges. The files to be deleted include:

Operating System	Files to be Removed Manually to Unregister an Existing Service Manager
Linux 6	• /etc/init.d/OracleGoldenGate • /etc/rc.d/*OracleGoldenGate • /etc/rc*.d/*OracleGoldenGate • /etc/oggInst.loc
Linux 7	/etc/systemd/system/ OracleGoldenGate.service

The following commands are executed to stop the Service Manager:

```
systemctl stop OracleGoldenGate
systemctl disable OracleGoldenGate *
```

Note

If the Service Manager is not registered as a service (with or without the integration with XAG), OGGCA stops the Service Manager deployment, otherwise, a script called `unregisterServiceManager` is created, and when executed by the user, it runs the `systemctl` commands and deletes the mentioned files.

3.2.3.2.2 How to Remove a Deployment: Silent Mode

You can remove a deployment silently using the Oracle GoldenGate Configuration Assistant (oggca) from the Oracle GoldenGate Home bin directory.

By removing a deployment, you can delete various components of the deployment, including, Extracts, Replicats, paths, and configuration files. However, the Service Manager is not deleted.

To remove a deployment silently:

Note

If the Service Manager is registered as a system service, removing a deployment silently will not unregister the service.

1. Ensure that you have a deployment response file. To get the deployment response file, run the OGGCA and then save the response file.
2. Update the following lines within the deployment response file:

```
CONFIGURATION_OPTION=REMOVE
ADMINISTRATOR_PASSWORD=*****
CREATE_NEW_SERVICEMANAGER=false
DEPLOYMENT_NAME=deployment_name
REMOVE_DEPLOYMENT_FROM_DISK=true
```

In case of multiple deployments, you must specify the deployment name using the DEPLOYMENT_NAME field. You can use the REMOVE_DEPLOYMENT_FROM_DISK option to remove physical files and folders associated with deployment.

3. Run the OGGCA program from the following location using the `-silent` and `-responseFile` options. Providing the exact path to the deployment response is needed.

```
$OGG_HOME/bin/oggca.sh -silent -responseFile
path_to_response_file/response_file.rsp
```

Example:

```
$OGG_HOME/bin/oggca.sh -silent -responseFile
/home/oracle/software/ogg_deployment.rsp
```

4

Get Started

- [Getting Started with Oracle GoldenGate for Distributed Applications and Analytics](#)
You can use Oracle GoldenGate for Distributed Applications and Analytics Microservices Architecture (MA) to configure and manage your data replication using an HTML user interface. The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) MA comprises the following components: Service Manager, Administration Service, Distribution Service, Receiver Service, Performance Metrics Service, and Admin Client.

4.1 Getting Started with Oracle GoldenGate for Distributed Applications and Analytics

You can use Oracle GoldenGate for Distributed Applications and Analytics Microservices Architecture (MA) to configure and manage your data replication using an HTML user interface. The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) MA comprises the following components: Service Manager, Administration Service, Distribution Service, Receiver Service, Performance Metrics Service, and Admin Client.

For more information about the Oracle GoldenGate MA components, see [Components of Oracle GoldenGate Microservices Architecture](#).

This topic lists the various tasks that you need to perform to set up GG for DAA integrations with cloud storage, message streaming, cloud warehouse, NoSQL and caching technologies.

- [Working With Deployments](#)
Adding deployments is the first task in the process of setting up a data replication platform. Deployments are managed from the Service Manager.
- [About Oracle GoldenGate Properties Files](#)
- [Using the Admin Client](#)
Admin Client is a command line utility (similar to the classic GGSCI utility). It uses the REST API published by the Microservices Servers to accomplish control and configuration tasks in an Oracle GoldenGate deployment.
- [Controlling Oracle GoldenGate \(Microservices Architecture\) Processes](#)
The standard way to control Oracle GoldenGate (MA) processes is through the Admin Client.

4.1.1 Working With Deployments

Adding deployments is the first task in the process of setting up a data replication platform. Deployments are managed from the Service Manager.

For more information about installing and deploying Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA), see [Installing Oracle GoldenGate for Distributed Applications and Analytics](#).

After you log into your Service Manager instance, you can create deployments or edit existing ones. You can work with multiple deployments from a single Service Manager instance. For more information about working with deployments, see [Working with Service Manager](#) in *Oracle GoldenGate Microservices Architecture Documentation* guide.

4.1.2 About Oracle GoldenGate Properties Files

There are two Oracle GoldenGate properties files required to run the Oracle GoldenGate Java Deliver user exit (alternatively called the Oracle GoldenGate Java Adapter). It is the Oracle GoldenGate Java Delivery that hosts Java integrations including the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) integrations. A Replicat properties file is required in order to run either process. The required naming convention for the Replicat file name is the *process_name.prm*. The exit syntax in the Replicat properties file provides the name and location of the Java Adapter properties file. It is the Java Adapter properties file that contains the configuration properties for the Java adapter include GG for DAA integrations. The Replicat and Java Adapters properties files are required to run GG for DAA integrations.

Alternatively the Java Adapters properties can be resolved using the default syntax, *process_name.properties*. If you use the default naming for the Java Adapter properties file then the name of the Java Adapter properties file can be omitted from the Replicat properties file.

Samples of the properties files for Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) integrations can be found in the subdirectories of the following directory:

GoldenGate_install_dir/AdapterExamples/big-data

- [Parameter Files](#)

Most of the Oracle GoldenGate functionality is controlled by the parameters specified in parameter files. A parameter file is a plain text file that is read by an associated Oracle GoldenGate process. Oracle GoldenGate uses two types of parameter files: a GLOBALS file and runtime parameter files.

4.1.2.1 Parameter Files

Most of the Oracle GoldenGate functionality is controlled by the parameters specified in parameter files. A parameter file is a plain text file that is read by an associated Oracle GoldenGate process. Oracle GoldenGate uses two types of parameter files: a GLOBALS file and runtime parameter files.

4.1.3 Using the Admin Client

Admin Client is a command line utility (similar to the classic GGSCI utility). It uses the REST API published by the Microservices Servers to accomplish control and configuration tasks in an Oracle GoldenGate deployment.

For more information about working with the Admin Client, see [Using the Admin Client](#) in the *Administering Oracle GoldenGate* guide.

4.1.4 Controlling Oracle GoldenGate (Microservices Architecture) Processes

The standard way to control Oracle GoldenGate (MA) processes is through the Admin Client.

Typically, the first time that Oracle GoldenGate processes are started in a production setting is during the initial synchronization process (also called instantiation process). However, you need to stop and start the processes at various points as needed to perform maintenance, upgrades, troubleshooting, or other tasks.

5

Upgrade

- [Upgrading Oracle GoldenGate for Distributed Applications and Analytics](#)
For Microservices, the earliest version that can be upgraded from is Oracle GoldenGate 26ai. As a best practice, perform a minimal upgrade first, so that you can troubleshoot more easily in case of any issue. After the environment is upgraded successfully, you can implement the new functionality.

5.1 Upgrading Oracle GoldenGate for Distributed Applications and Analytics

For Microservices, the earliest version that can be upgraded from is Oracle GoldenGate 26ai. As a best practice, perform a minimal upgrade first, so that you can troubleshoot more easily in case of any issue. After the environment is upgraded successfully, you can implement the new functionality.

This chapter describes how to upgrade Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) from the previous releases to the current release.

The pre-upgrade requirements are as follows:

- Stop all Oracle GoldenGate processes.
- Start Oracle GoldenGate.

Topics:

- [Obtaining the Oracle GoldenGate Distribution](#)
- [Scope of Upgrade](#)
Even though you may only upgrade the source or target, rather than both, all processes are involved in upgrade. All processes must be stopped in the correct order for the upgrade, regardless of which component you upgrade, and the trails must be processed until empty.
- [Upgrading Oracle GoldenGate for Distributed Applications and Analytics – GUI Based](#)

5.1.1 Obtaining the Oracle GoldenGate Distribution

To obtain Oracle GoldenGate:

1. Go to edelivery.oracle.com. For more information, see [My Oracle Support Banner Oracle GoldenGate -- Oracle RDBMS Server Recommended Patches \(Doc ID 1557031.1\)](#). To access Oracle Technology Network (OTN), go to <https://www.oracle.com/integration/goldengate/>
2. Find the Oracle GoldenGate 26ai release and download the ZIP file onto your system.

For more information about locating and downloading Oracle Fusion Middleware products, see the [Oracle® Fusion Middleware Download, Installation, and Configuration Readme Files](#) on OTN.

5.1.2 Scope of Upgrade

Even though you may only upgrade the source or target, rather than both, all processes are involved in upgrade. All processes must be stopped in the correct order for the upgrade, regardless of which component you upgrade, and the trails must be processed until empty.

Before you start the upgrade, review the information about upgrading Extract and Replicat.

Oracle recommends that you begin your upgrade with the target rather than the source to avoid the necessity of adjusting the trail file format.

- [Replicat Upgrade Considerations](#)
All Replicat installations should be upgraded at the same time. It is critical to ensure that all trails leading to all Replicat groups on all target systems are processed until empty, according to the upgrade instructions.

5.1.2.1 Replicat Upgrade Considerations

All Replicat installations should be upgraded at the same time. It is critical to ensure that all trails leading to all Replicat groups on all target systems are processed until empty, according to the upgrade instructions.

Before you start the upgrade, review the information about upgrading Extract and Replicat.

Oracle recommends that you begin your upgrade with the target rather than the source to avoid the necessity of adjusting the trail file format.

5.1.3 Upgrading Oracle GoldenGate for Distributed Applications and Analytics – GUI Based

To obtain the Oracle GoldenGate installation software and set up the directories for upgrade:

1. Download the Oracle GoldenGate DAA for 26ai software from the Oracle Technology Network or eDelivery.
2. Upload the Oracle GoldenGate Microservices 21c software to a staging location on the server where a previous release of Oracle GoldenGate Microservices exists.
3. Unzip Oracle GoldenGate Microservices 21c software in the staging location.

```
$ cd /tmp $ unzip  
./fbo_ggs_Linux_x64_services_shiphome.zip
```
4. Untar the tar file that gets created after the unzip command: `tar -xvf ggs_Linux_x64_Oracle_64bit.tar`
5. Move into the unzipped files and execute the `runInstaller` command.

```
$ cd ./fbo_ggs_Linux_x64_services_shiphome/Disk1  
$ ./runInstaller
```
6. For Software Location, specify where the new Oracle GoldenGate home is located. This is not the same location as the current Oracle GoldenGate home. Click **Next**.
7. Click **Install** to begin installing the new GG for DAA. When the installation is done, click **Close**.

At this point, you should have two GG for DAA home directories: one for your old home (21c) and a new home (26ai).

8. Verify the current version of Oracle GoldenGate Home through Service Manager.
 - a. Login to the Service Manager:

`http://host:servicemanager_port`
 - b. Review the deployment section for your current Oracle GoldenGate home location.
9. Update the Service Manager and the deployments with the location of the new Oracle GoldenGate home.
 - a. Click **Service Manager**, then **Deployment name**.
 - b. Next to the deployment details, click the pencil icon to display the dialog box to edit the Oracle GoldenGate home.
 - c. Update the Oracle GoldenGate home with the complete path to the new Oracle GoldenGate home. Also update the following, if required:

`LD_LIBRARY_PATH`
 - d. Click **Apply**.
 - e. Confirm that the Oracle GoldenGate home has been updated.
 - f. Stop all Extracts, Replicats, and Distribution paths.
 - g. Use the action button to restart Service Manager or Deployment.

 Note

You can confirm that the Oracle GoldenGate home was updated by looking at the process from the operating system for Service Manager. The Service Manager process should be running from the new Oracle GoldenGate home.

10. To upgrade the associated deployments, follow the same steps for Service Manager after ensuring that all the Extract and Replicat processes in that deployment have been stopped.

6

Secure

- [Security](#)

6.1 Security

Learn about how to protect and secure your Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) environment, including details on TLS support, authentication and authorization, trail file encryption options, streaming protocols, and other features that form the Oracle GoldenGate security framework.

Oracle GoldenGate security documentation is divided into Oracle GoldenGate Security Features and Oracle GoldenGate Security Features: Implementation.

[Oracle GoldenGate Security Features](#) section describes the of security standards applied with Oracle GoldenGate. This section is aimed to help Chief Security Officers (CSOs) learn about Oracle GoldenGate security features and standards.

[Oracle GoldenGate Security Feature: Implementation](#) section contains steps to implement various advanced security features available with Oracle GoldenGate. This section is aimed to assist DBAs and Oracle GoldenGate administrators implement security features available within Oracle GoldenGate.

7

Configure

- [Configuring Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Logging](#)
Logging is essential to troubleshooting Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) integrations with GG for DAA targets.
- [Configuring Logging](#)

7.1 Configuring Oracle GoldenGate for Distributed Applications and Analytics

This topic describes how to configure GG for DAA handlers.

- [Running with Replicat](#)
You need to review before configuring a replicat process in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).
- [About Schema Evolution and Metadata Change Events](#)
- [About Configuration Property CDATA\[\] Wrapping](#)
- [Using Regular Expression Search and Replace](#)
You can perform more powerful search and replace operations of both schema data (catalog names, schema names, table names, and column names) and column value data, which are separately configured. Regular expressions (regex) are characters that customize a search string through pattern matching.
- [Scaling Oracle GoldenGate for Distributed Applications and Analytics Delivery](#)
- [Coordinated Apply Support](#)
- [Configuring Cluster High Availability](#)
Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) doesn't have built-in high availability functionality. You need to use a standard cluster software's high availability capability to provide the high availability functionality.
- [Using Identities in Oracle GoldenGate Credential Store](#)
The Oracle GoldenGate credential store manages user IDs and their encrypted passwords (together known as credentials) that are used by Oracle GoldenGate processes to interact with the local database. The credential store eliminates the need to specify user names and clear-text passwords in the Oracle GoldenGate parameter files.

7.1.1 Running with Replicat

You need to review before configuring a replicat process in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

This topic explains how to run the Java Adapter with the Oracle GoldenGate Replicat process.

- [Replicat Grouping](#)
- [About Replicat Checkpointing](#)

- [About Initial Load Support](#)
- [About the Unsupported Replicat Features](#)
- [How the Mapping Functionality Works](#)

7.1.1.1 Replicat Grouping

The Replicat process provides the Replicat configuration property, `GROUPTRANSOPS`, to control transaction grouping. By default, the Replicat process implements transaction grouping of 1000 source transactions into a single target transaction. If you want to turn off transaction grouping then the `GROUPTRANSOPS` Replicat property should be set to 1.

7.1.1.2 About Replicat Checkpointing

In addition to the Replicat checkpoint file, `.cpr`, an additional checkpoint file, `dirchk/group.cpj`, is created that contains information similar to `CHECKPOINTTABLE` in Replicat for the database.

7.1.1.3 About Initial Load Support

Replicat can already read trail files that come from both the online capture and initial load processes that write to a set of trail files. In addition, Replicat can also be configured to support the delivery of the special run initial load process using `RMTTASK` specification in the Extract parameter file. For more details about configuring the direct load, see [Initial Load Extract](#).

Note

The `SOURCEDB` or `DBLOGIN` parameter specifications vary depending on your source database.

7.1.1.4 About the Unsupported Replicat Features

The following Replicat features are not supported in this release:

- `BATCHSQL`
- `SQLEXEC`
- Stored procedure
- Conflict resolution and detection (CDR)

7.1.1.5 How the Mapping Functionality Works

The Oracle GoldenGate Replicat process supports mapping functionality to custom target schemas. You must use the Metadata Provider functionality to define a target schema or schemas, and then use the standard Replicat mapping syntax in the Replicat configuration file to define the mapping.

7.1.2 About Schema Evolution and Metadata Change Events

The Metadata in trail is a feature that allows seamless runtime handling of metadata change events by Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA), including schema evolution and schema propagation to GG for DAA target applications. The

NO_OBJECTDEFS is a sub-parameter of the Extract and Replicat EXTTRAIL and RMTTRAIL parameters that lets you suppress the important metadata in trail feature and revert to using a static metadata definition.

The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) Handlers and Formatters provide functionality to take action when a metadata change event is encountered. The ability to take action in the case of metadata change events depends on the metadata change events being available in the source trail file. Oracle GoldenGate supports metadata in trail and the propagation of DDL data from a source Oracle Database. If the source trail file does not have metadata in trail and DDL data (metadata change events) then it is not possible for GG for DAA to provide and metadata change event handling.

7.1.3 About Configuration Property CDATA[] Wrapping

The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) Handlers and Formatters support the configuration of many parameters in the Java properties file, the value of which may be interpreted as white space. The configuration handling of the Java Adapter trims white space from configuration values from the Java configuration file. This behavior of trimming whitespace may be desirable for some configuration values and undesirable for other configuration values. Alternatively, you can wrap white space values inside of special syntax to preserve the white space for selected configuration variables. GG for DAA borrows the XML syntax of CDATA[] to preserve white space. Values that would be considered to be white space can be wrapped inside of CDATA[].

The following is an example attempting to set a new-line delimiter for the Delimited Text Formatter:

```
gg.handler.{name}.format.lineDelimiter=\n
```

This configuration will not be successful. The new-line character is interpreted as white space and will be trimmed from the configuration value. Therefore the `gg.handler` setting effectively results in the line delimiter being set to an empty string.

In order to preserve the configuration of the new-line character simply wrap the character in the CDATA[] wrapper as follows:

```
gg.handler.{name}.format.lineDelimiter=CDATA[\n]
```

Configuring the property with the CDATA[] wrapping preserves the white space and the line delimiter will then be a new-line character.

7.1.4 Using Regular Expression Search and Replace

You can perform more powerful search and replace operations of both schema data (catalog names, schema names, table names, and column names) and column value data, which are separately configured. Regular expressions (regex) are characters that customize a search string through pattern matching.

You can match a string against a pattern or extract parts of the match. Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) uses the standard Oracle Java regular expressions package, `java.util.regex`, see Regular Expressions in [The Single UNIX Specification, Version 4](#).

- [Using Schema Data Replace](#)
- [Using Content Data Replace](#)

7.1.4.1 Using Schema Data Replace

You can replace schema data using the `gg.schemareplaceregex` and `gg.schemareplacestring` properties. Use `gg.schemareplaceregex` to set a regular expression, and then use it to search catalog names, schema names, table names, and column names for corresponding matches. Matches are then replaced with the content of the `gg.schemareplacestring` value. The default value of `gg.schemareplacestring` is an empty string or `""`.

For example, some system table names start with a dollar sign like `$mytable`. You may want to replicate these tables even though most technologies do not allow dollar signs in table names. To remove the dollar sign, you could configure the following replace strings:

```
gg.schemareplaceregex=[$]  
gg.schemareplacestring=
```

The resulting example of searched and replaced table name is `mytable`. These properties also support `CDATA[]` wrapping to preserve whitespace in the value of configuration values. So the equivalent of the preceding example using `CDATA[]` wrapping use is:

```
gg.schemareplaceregex=CDATA[[  
gg.schemareplacestring=CDATA[
```

The schema search and replace functionality supports using multiple search regular expressions and replacements strings using the following configuration syntax:

```
gg.schemareplaceregex=some_regex  
gg.schemareplacestring=some_value  
gg.schemareplaceregex1=some_regex  
gg.schemareplacestring1=some_value  
gg.schemareplaceregex2=some_regex  
gg.schemareplacestring2=some_value
```

7.1.4.2 Using Content Data Replace

You can replace content data using the `gg.contentreplaceregex` and `gg.contentreplacestring` properties to search the column values using the configured regular expression and replace matches with the replacement string. For example, this is useful to replace line feed characters in column values. If the delimited text formatter is used then line feeds occurring in the data will be incorrectly interpreted as line delimiters by analytic tools.

You can configure *n* number of content replacement regex search values. The regex search and replacements are done in the order of configuration. Configured values must follow a given order as follows:

```
gg.contentreplaceregex=some_regex  
gg.contentreplacestring=some_value  
gg.contentreplaceregex1=some_regex  
gg.contentreplacestring1=some_value  
gg.contentreplaceregex2=some_regex  
gg.contentreplacestring2=some_value
```

Configuring a subscript of 3 without a subscript of 2 would cause the subscript 3 configuration to be ignored.

NOT_SUPPORTED

Regular express searches and replacements require computer processing and can reduce the performance of the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) process.

To replace line feeds with a blank character you could use the following property configurations:

```
gg.contentreplaceregex=[\n]
gg.contentreplacestring=CDATA[ ]
```

This changes the column value from:

```
this is
me
```

to :

```
this is me
```

Both values support CDATA wrapping. The second value must be wrapped in a CDATA[] wrapper because a single blank space will be interpreted as whitespace and trimmed by the GG for DAA configuration layer. In addition, you can configure multiple search a replace strings. For example, you may also want to trim leading and trailing white space out of column values in addition to trimming line feeds from:

```
^\\s+|\\s+$
```

```
gg.contentreplaceregex1=^\\s+|\\s+$
gg.contentreplacestring1=CDATA[ ]
```

7.1.5 Scaling Oracle GoldenGate for Distributed Applications and Analytics Delivery

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) supports breaking down the source trail files into either multiple Replicat processes or by using Coordinated Delivery to instantiate multiple Java Adapter instances inside a single Replicat process to improve throughput. This allows you to scale GG for DAA delivery.

There are some cases where the throughput to GG for DAA integration targets is not sufficient to meet your service level agreements even after you have tuned your handler for maximum performance. When this occurs, you can configure parallel processing and delivery to your targets using one of the following methods:

- Multiple Replicat processes can be configured to read data from the same source trail files. Each of these Replicat processes are configured to process a subset of the data in the source trail files so that all of the processes collectively process the source trail files in their entirety. There is no coordination between the separate Replicat processes using this solution.
- Oracle GoldenGate Coordinated Delivery can be used to parallelize processing the data from the source trail files within a single Replicat process. This solution involves breaking the trail files down into logical subsets for which each configured subset is processed by a different delivery thread. For more information about Co-ordinated Replicat, see [About Coordinated Replicat](#) in the *Oracle GoldenGate Microservices Architecture Documentation*.

With either method, you can split the data into parallel processing for improved throughput. Oracle recommends breaking the data down in one of the following two ways:

- **Splitting Source Data By Source Table** – Data is divided into subsections by source table. For example, Replicat process 1 might handle source tables table1 and table 2, while Replicat process 2 might handle data for source tables table3 and table2. Data is split for source table and the individual table data is not subdivided.
- **Splitting Source Table Data into Sub Streams** – Data from source tables is split. For example, Replicat process 1 might handle half of the range of data from source table1, while Replicat process 2 might handler the other half of the data from source table1.
- If you are using Coordinated Replicat, please make sure that you add TARGETDB LIBFILE libggjava.so SET property=path_to_deployment_home/etc/conf/ogg/your_replicat_name.properties.

Additional limitations:

- Parallel apply is *not* supported.
- The BATCHSQL parameter not supported.

Example 7-1 Scaling Support for the Oracle GoldenGate for Distributed Applications and Analytics Handlers

Handler Name	Splitting Source Data By Source Table	Splitting Source Table Data into Sub Streams
Cassandra	Supported	Supported when: • Required target tables in Cassandra are pre-created. • Metadata change events do not occur.
Elastic Search	Supported	Supported

Handler Name	Splitting Source Data By Source Table	Splitting Source Table Data into Sub Streams
HBase	Supported when all required HBase namespaces are pre-created in HBase.	Supported when: - All required HBase namespaces are pre-created in HBase. - All required HBase target tables are pre-created in HBase. Schema evolution is not an issue because HBase tables have no schema definitions so a source metadata change does not require any schema change in HBase. - The source data does not contain any truncate operations.
HDFS	Supported	Supported with some restrictions. - You must select a naming convention for generated HDFS files where the file names do not collide. Colliding HDFS file names results in a Replicatabend. When using coordinated apply it is suggested that you configure <code>\${groupName}</code> as part of the configuration for the <code>gg.handler.name.fileNameMappingTemplate</code> property. The <code>\${groupName}</code> template resolves to the Replicat name concatenated with the Replicat thread number, which provides unique naming per Replicat thread. For more information, see Coordinated Apply Support. - Schema propagation to HDFS and Hive integration is <i>not</i> currently supported.
JDBC	Supported	Supported
Kafka	Supported	Supported for formats that support schema propagation, such as Avro. This is less desirable due to multiple instances feeding the same schema information to the target.
Kafka Connect	Supported	Supported
Kinesis Streams	Supported	Supported
MongoDB	Supported	Supported

Handler Name	Splitting Source Data By Source Table	Splitting Source Table Data into Sub Streams
Java File Writer	Supported	Supported with the following restrictions: You must select a naming convention for generated files where the file names do not collide. Colliding file names may results in a Replicat abend and/or polluted data. When using coordinated apply it is suggested that you configure \$ {groupName} as part of the configuration for the <code>gg.handler.name.fileNameMappingTemplate</code> property . The \$ {groupName} template resolves to the Replicat name concatenated with the Replicat thread number, which provides unique naming per Replicat thread. For more information, see Coordinated Apply Support .

7.1.6 Coordinated Apply Support

Support Matrix for Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) Targets

Feature index

- Option1 - Discreet threads per table - Initial load
- Option2 - Shared threads per table - Initial load
- Option3 - Discreet threads per table - CDC
- Option4 - Shared threads per table - CDC

Target	Option1	Option2	Option3	Option4
Snowflake stage/merge	[✓]	[✓]	[✓]	*SNOW [✓]
BigQuery stage/merge	[✓]	[✓]	[✓]	*BQ [✓]
Databricks	[✓]	*DBX [✓]	[✓]	*DBX [✓]
ADW	[✓]	[✓]	[✓]	[✓]
Synapse	[✓]	[✓]	[✓]	*SYN[✓]
Redshift	[✓]	*RS[✓]	[✓]	*RS[✓]
OneLake	[✓]	[x]	[✓]	[x]
Iceberg	[✓]	*ICE[✓]	[✓]	*ICE[✓]
Snowflake streaming	[✓]	*TC [✓]	[✓]	*TC [✓]
BigQuery streaming	[✓]	*TC [✓]	[✓]	*TC [✓]

Target	Option1	Option2	Option3	Option4
Cassandra	[✓]	*TC [✓]	[✓]	*TC [✓]
HBase	[✓]	*HBASE [✓]	[✓]	*HBASE [✓]
Elasticsearch	[✓]	[✓]	[✓]	[✓]
Kafka	[✓]	[✓]	[✓]	[✓]
Oracle Streaming	[✓]	[✓]	[✓]	[✓]
Kafka Connect	[✓]	[✓]	[✓]	[✓]
MongoDB	[✓]	[✓]	[✓]	[✓]
AJD	[✓]	[✓]	[✓]	[✓]
CosmosDB	[✓]	[✓]	[✓]	[✓]
Kinesis	[✓]	[✓]	[✓]	[✓]
Kafka Rest Proxy	[✓]	[✓]	[✓]	[✓]
Oracle NoSQL	[✓]	*TC [✓]	[✓]	*TC [✓]
Redis	[✓]	*REDIS [✓]	[✓]	*REDIS [✓]
Google Pub/Sub	[✓]	[✓]	[✓]	[✓]
File Writer	[✓]	*FW [✓]	[✓]	*FW [✓]
Hadoop (HDFS)	[✓]	*FW [✓]	[✓]	*FW [✓]
JDBC	[✓]	[✓]	[✓]	[✓]

***SNOW**

- Option4 is supported for Snowflake but not recommended.
- MERGE statement holds a table lock until the transaction is committed. The locks times may hinder the performance of the replicat.

***BQ**

- Option4 is supported for BigQuery but not recommended.
- At times, replicat can ABEND with Query error: Could not serialize access to table due to concurrent update
- BigQuery jobs are retried, by default up to three times. We have a few configuration parameters related to this. For more information, see `gg.eventhandler.bq.retries` and `gg.eventhandler.bq.totalTimeout` in [BigQuery Event Handler Configuration](#).

***DBX**

- Option4 is supported for Databricks but not recommended.
- If the INSERT or MERGE DML modifies the same partition in a table (or if the table is unpartitioned), it could lead to `ConcurrentAppendException`. See <https://docs.databricks.com/en/optimizations/isolation-level.html#concurrentappendexception>

***SYN**

- Option 4 is supported. For Synapse, the isolation level of the transactional support is default to READ UNCOMMITTED. See: <https://learn.microsoft.com/en-us/sql/t-sql/statements/set-transaction-isolation-level-transact-sql?view=sql-server-ver16#arguments>
- If you modify the default isolation level, then there could be a performance impact and errors when using the MERGE statement.

***RS**

- Option 2 and Option 4 are supported for Redshift but provided the Redshift database's isolation level is set to SNAPSHOT.
- To enable co-ordinated apply for Redshift, ensure that the Redshift database's isolation level is set to SNAPSHOT.
- The Redshift SNAPSHOT ISOLATION option allows higher concurrency, where concurrent modifications to different rows in the same table can complete successfully. SQL:

```
ALTER DATABASE <sampledb> ISOLATION LEVEL SNAPSHOT;
```

*ICE

- Option 2 and Option 4 are supported for most Iceberg catalogs except the Hadoop catalog.

*TC

- Not recommended if auto table creation and/or altering is enabled. Threads of execution may conflict during table creation and/or altering.

*HBASE

- Precreate namespaces and target tables in HBase.

*REDIS

- Disable automatic index creation in the configuration otherwise threads of execution may conflict with each other when creating indexes.

*FW

- Configure a file naming template which ensures that file names are unique across threads of execution. A good strategy is to use \${groupname} in the template.

7.1.7 Configuring Cluster High Availability

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) doesn't have built-in high availability functionality. You need to use a standard cluster software's high availability capability to provide the high availability functionality.

You can configure a high availability scenario on a cluster so that if the leader instance of (GG for DAA) on machine fails, another GG for DAA instance could be started on another machine to resume where the failed instance left off.

If you manually configure your instances to share common GG for DAA and Oracle GoldenGate files using a shared disk architecture you can create a fail over situation. For a cluster installation, these files would need to be accessible from all machines and accessible in the same location.

The configuration files that must be shared are:

- replicat.prm
- Handler properties file.
- Additional properties files required by the specific adapter. This depends on the target handler in use. For example, Kafka would be a producer properties file.
- Additional schema files you've generated. For example, Avro schema files generated in the `dirdef` directory.
- File Writer Handler generated files on your local file system at a configured path. Also, the File Writer Handler state file in the `dirsta` directory.

- Any `log4j.properties` or `logback.properties` files in use.

Checkpoint files must be shared for the ability to resume processing:

- Your Replicat checkpoint file (`*.cpr`).
- Your adapter checkpoint file (`*.cpj`).

7.1.8 Using Identities in Oracle GoldenGate Credential Store

The Oracle GoldenGate credential store manages user IDs and their encrypted passwords (together known as credentials) that are used by Oracle GoldenGate processes to interact with the local database. The credential store eliminates the need to specify user names and clear-text passwords in the Oracle GoldenGate parameter files.

An optional alias can be used in the parameter file instead of the user ID to map to a userid and password pair in the credential store. The credential store is implemented as an auto login wallet within the Oracle Credential Store Framework (CSF). The use of an LDAP directory is not supported for the Oracle GoldenGate credential store. The auto login wallet supports automated restarts of Oracle GoldenGate processes without requiring human intervention to supply the necessary passwords.

In Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA), you specify the alias and domain in the property file not the actual user ID or password. User credentials are maintained in secure wallet storage.

- [Creating a Credential Store](#)
- [Adding Users to a Credential Store](#)
- [Configuring Properties to Access the Credential Store](#)

7.1.8.1 Creating a Credential Store

You can create a credential store for your Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) environment.

Run the `GGSCI ADD CREDENTIALSTORE` command to create a file called `cwallet.sso` in the `dircrd/` subdirectory of your Oracle GoldenGate installation directory (the default).

You can the location of the credential store (`cwallet.sso` file by specifying the desired location with the `CREDENTIALSTORELOCATION` parameter in the `GLOBALS` file.

Note

Only one credential store can be used for each Oracle GoldenGate instance.

7.1.8.2 Adding Users to a Credential Store

After you create a credential store for your Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) environment, you can added users to the store.

Run the `GGSCI ALTER CREDENTIALSTORE ADD USER userid PASSWORD password [ALIAS alias] [DOMAIN domain]` command to create each user, where:

- *userid* is the user name. Only one instance of a user name can exist in the credential store unless the `ALIAS` or `DOMAIN` option is used.

- *password* is the user's password. The password is echoed (not obfuscated) when this option is used. If this option is omitted, the command prompts for the password, which is obfuscated as it is typed (recommended because it is more secure).
- *alias* is an alias for the user name. The alias substitutes for the credential in parameters and commands where a login credential is required. If the ALIAS option is omitted, the alias defaults to the user name.

For example:

```
ALTER CREDENTIALSTORE ADD USER scott PASSWORD tiger ALIAS scsm2 domain
ggadapters
```

7.1.8.3 Configuring Properties to Access the Credential Store

The Oracle GoldenGate Java Adapter properties file requires specific syntax to resolve user name and password entries in the Credential Store at runtime. For resolving a user name the syntax is the following:

```
ORACLEWALLETUSERNAME[alias domain_name]
```

For resolving a password the syntax required is the following:

```
ORACLEWALLETPASSWORD[alias domain_name]
```

The following example illustrate how to configure a Credential Store entry with an alias of myalias and a domain of mydomain.

Note

With HDFS Hive JDBC the user name and password is encrypted.

Oracle Wallet integration only works for configuration properties which contain the string username or password. For example:

```
gg.handler.hdfs.hiveJdbcUsername=ORACLEWALLETUSERNAME[myalias mydomain]
gg.handler.hdfs.hiveJdbcPassword=ORACLEWALLETPASSWORD[myalias mydomain]
```

ORACLEWALLETUSERNAME and ORACLEWALLETPASSWORD can be used in the Extract (similar to Replicat) in JMS handler as well. For example:

```
gg.handler.<name>.user=ORACLEWALLETUSERNAME[JMS_USR JMS_PWD]
gg.handler.<name>.password=ORACLEWALLETPASSWORD[JMS_USR JMS_PWD]
```

Consider the user name and password entries as accessible values in the Credential Store. Any configuration property resolved in the Java Adapter layer (not accessed in the C user exit layer) can be resolved from the Credential Store. This allows you more flexibility to be creative in how you protect sensitive configuration entries.

7.2 Logging

Logging is essential to troubleshooting Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) integrations with GG for DAA targets.

This topic details how GG for DAA integration log and the best practices for logging.

- [About Replicat Process Logging](#)
- [About Java Layer Logging](#)
- [About SQL Statement Logging](#)

7.2.1 About Replicat Process Logging

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) integrations leverage the Java Delivery functionality. In this setup, either a Oracle GoldenGate Replicat process loads a user exit shared library. This shared library then loads a Java virtual machine to thereby interface with targets providing a Java interface. So the flow of data is as follows:

Replicat Process —> User Exit —> Java Layer

It is important that all layers log correctly so that users can review the logs to troubleshoot new installations and integrations. Additionally, if you have a problem that requires contacting Oracle Support, the log files are a key piece of information to be provided to Oracle Support so that the problem can be efficiently resolved.

A running Replicat process creates or appends log files into the *GoldenGate_Home/dirrpt* directory that adheres to the following naming convention: *process_name.rpt*. If a problem is encountered when deploying a new Oracle GoldenGate process, this is likely the first log file to examine for problems. The Java layer is critical for integrations with GG for DAA applications.

7.2.2 About Java Layer Logging

The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) product provides flexibility for logging from the Java layer. The recommended best practice is to use Log4j logging to log from the Java layer. Enabling simple Log4j logging requires the setting of two configuration values in the Java Adapters configuration file.

```
gg.log=log4j  
gg.log.level=INFO
```

These gg.log settings will result in a Log4j file to be created in the *GoldenGate_Home/dirrpt* directory that adheres to this naming convention, *{GROUPNAME}.log*. The supported Log4j log levels are in the following list in order of increasing logging granularity.

- OFF
- FATAL
- ERROR
- WARN
- INFO
- DEBUG
- TRACE

Selection of a logging level will include all of the coarser logging levels as well (that is, selection of WARN means that log messages of FATAL, ERROR and WARN will be written to the log file). The Log4j logging can additionally be controlled by separate Log4j properties files. These separate Log4j properties files can be enabled by editing the `bootoptions` property in the Java Adapter Properties file. These three example Log4j properties files are included with the installation and are included in the classpath:

```
log4j-default.properties
log4j-debug.properties
log4j-trace.properties
```

You can modify the `bootoptions` in any of the files as follows:

```
javawriter.bootoptions=-Xmx512m -Xms64m -Djava.class.path=.:ggjava/ggjava.jar -
Dlog4j.configurationFile=samplelog4j.properties
```

You can use your own customized Log4j properties file to control logging. The customized Log4j properties file must be available in the Java classpath so that it can be located and loaded by the JVM. The contents of a sample custom Log4j properties file is the following:

```
# Root logger option
log4j.rootLogger=INFO, file

# Direct log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.File=sample.log
log4j.appender.file.MaxFileSize=1GB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```

There are two important requirements when you use a custom Log4j properties file. First, the path to the custom Log4j properties file must be included in the `javawriter.bootoptions` property. Logging initializes immediately when the JVM is initialized while the contents of the `gg.classpath` property is actually appended to the classloader after the logging is initialized. Second, the classpath to correctly load a properties file must be the directory containing the properties file without wildcards appended.

7.2.3 About SQL Statement Logging

Oracle GoldenGate for Distributed Applications and Analytics now includes the ability to log SQL statements for stage and merge targets into a file. This feature is designed to assist with debugging by providing insight into the execution of SQL statements processed by Replicat.

By default, SQL logging is set to the INFO level. The generated log files are stored in the `OGG_DIRLOG` directory. Each SQL log entry includes a timestamp to make it easier to trace the execution sequence. Note that `SHOWSYNTAX` is not supported for coordinated Replicat groups.

This functionality is intended specifically for debugging purposes. It is recommended to use it when you need to verify the SQL syntax generated by Replicat as part of troubleshooting or validation.

- [Configuring SQL Statement Logging](#)

7.2.3.1 Configuring SQL Statement Logging

Parameter file

SHOWSYNTAX _NOCHECKTTY

Property file

- `gg.log.sql.filename` (Optional)
- `gg.log.sql.level=info` (Optional)

Note

Including this, the configuration in the properties file is optional. If not specified, Replicat will use the default settings, with the log file named after the Replicat process name.

For more information about the SHOWSYNTAX parameter, see [SHOWSYNTAX](#) in the *Reference for Oracle GoldenGate*.

7.3 Configuring Logging

- [Oracle GoldenGate Java Adapter Default Logging](#)
- [Recommended Logging Settings](#)

7.3.1 Oracle GoldenGate Java Adapter Default Logging

- [Default Logging Setup](#)
- [Log File Name](#)
- [Changing Logging Level](#)

7.3.1.1 Default Logging Setup

Logging is enabled by default for the Oracle GoldenGate for BigData. The logging implementation is log4j. By default, logging is enabled at the info level.

7.3.1.2 Log File Name

The log output file is created in the standard report directory. The name of the log file includes the replicat group name and has an extension of log.

If the Oracle GoldenGate Replicat process group name is JAVAUE, then the log file name in the report directory is: JAVAUE.log.

7.3.1.3 Changing Logging Level

To change the recommended log4j logging level, add the configuration shown in the following example to the Java Adapter Properties file:

```
gg.log.level=error
```

You can set the `gg.log.level` to `none`, `error`, `warn`, `info`, `debug`, or `trace`. The default log level is `info`. Oracle recommends the `debug` and `trace` log levels only for troubleshooting as these settings can adversely impact the performance.

7.3.2 Recommended Logging Settings

Oracle recommends that you use log4j logging instead of the JDK default for unified logging for the Java user exit. Using log4j provides unified logging for the Java module when running with the Oracle GoldenGate Replicat process.

- [Changing to the Recommended Logging Type](#)

7.3.2.1 Changing to the Recommended Logging Type

To change the recommended log4j logging implementation, add the configuration shown in the following example to the Java Adapter Properties file.

```
gg.log=log4j  
gg.log.level=info
```

The `gg.log.level` can be set to `none`, `error`, `warn`, `info`, `debug`, or `trace`. The default log level is `info`. The `debug` and `trace` log levels are only recommended for troubleshooting as these settings can adversely affect performance.

The result is that a log file for the Java module will be created in the `dirrpt` directory with the following naming convention:

```
<process name>_<log level>log4j.log
```

Therefore if the Oracle GoldenGate Replicat process is called `javaue`, and the `gg.log.level` is set to `debug`, the resulting log file name is:

```
javaue_debug_log4j.log
```


8

Quickstarts

This article will get help you in quickly getting started with the following tasks in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

- [QuickStarts: Prerequisites](#)
- [Realtime Data Ingestion into Snowflake with Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Realtime Parquet Ingestion into Google Cloud Storage with GoldenGate for Distributed Applications and Analytics 23.8 and later](#)
- [Realtime Parquet Ingestion into Google Cloud Storage with Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Realtime Parquet Ingestion into AWS S3 Buckets with Oracle GoldenGate for Distributed Applications and Analytics 23.8 and later](#)
- [Realtime Parquet Ingestion into AWS S3 Buckets with Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Realtime Parquet Ingestion into Azure Data Lake Storage with Oracle GoldenGate for Distributed Applications and Analytics 23.8 and later](#)
- [Realtime Parquet Ingestion into Azure Data Lake Storage with Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Realtime Parquet Ingestion into OCI Object Storage with Oracle GoldenGate for Distributed Applications and Analytics 23.8 and later](#)
- [Realtime Parquet Ingestion into OCI Object Storage with Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Realtime Message Ingestion to OCI Streaming with Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Realtime Message Ingestion to Azure Event Hubs with Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Realtime Data Ingestion into GCP BigQuery with Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Realtime Message Ingestion to Google Pub/Sub with Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Realtime Message Ingestion to Apache Kafka with Oracle GoldenGate for Distributed Applications and Analytics](#)
- [Realtime Data Ingestion into Azure Databricks \(unity catalog enabled\) with GoldenGate for DAA](#)
- [Realtime Data Ingestion into Oracle AI Data Platform with Oracle GoldenGate for DAA](#)

8.1 QuickStarts: Prerequisites

- It is assumed that you've installed Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) in your environment or from Oracle Cloud Infrastructure

Marketplace. See [Installing Oracle GoldenGate MA for Distributed Applications and Analytics Using the UI](#).

- It is assumed that you have configured an Oracle GoldenGate extract, which is up and running and the trails are being sent to GG for DAA Deployment. See [Add Extracts](#) in *Oracle GoldenGate Microservices Documentation*.
- Get familiar with Oracle GoldenGate Microservices by watching this video: [Introduction to Oracle GoldenGate Microservices](#).

8.2 Realtime Data Ingestion into Snowflake with Oracle GoldenGate for Distributed Applications and Analytics

Overview

This Quickstart covers a step-by-step process showing how to ingest real-time data into Snowflake with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

Snowflake is a cloud-based data warehousing platform that provides a fully managed service for storing, processing, and analysing data.

GG for DAA Snowflake handler uses the stage and merge data flow. It supports both external and internal staging options.

In stage and merge, the change data is staged in a temporary location in microbatches and eventually merged into the target table. If external staging is selected, storage location is cloud storage service of the cloud provider (Azure Storage for Azure, S3 for AWS and Cloud Storage for GCP). If internal staging is selected, then external storage location will not be needed.

All replication process is automatically handled by [Snowflake Stage and Merge Handler](#) handler.

This quickstart covers the steps for internal staging. External staging requires a storage service and a Snowflake storage integration.

- [Prerequisites for Internal Staging](#)
- [Install Dependency Files](#)
- [Create a Credential Store Entry](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.2.1 Prerequisites for Internal Staging

- A Snowflake account.
- Target schema and database in Snowflake
- JDBC URL for Snowflake access.
- Snowflake username/ password

In this Quickstart, a sample trail file (named tr), which is shipped with GG for DAA is used. The sample trail file is located at GG_HOME/opt/AdapterExamples/trail/ in your GG for DAA instance.

GG for DAA creates the target Snowflake tables automatically.

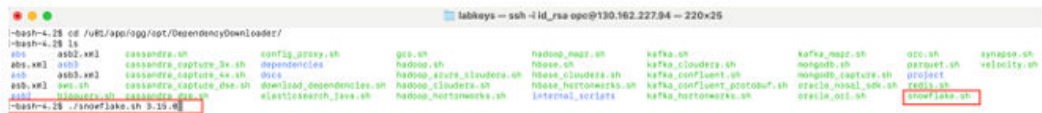
8.2.2 Install Dependency Files

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) uses Java SDK provided by Snowflake. You can download the SDKs using [Dependency Downloader](#) utility shipped with GG for DAA. Dependency downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

To install the required dependency files:

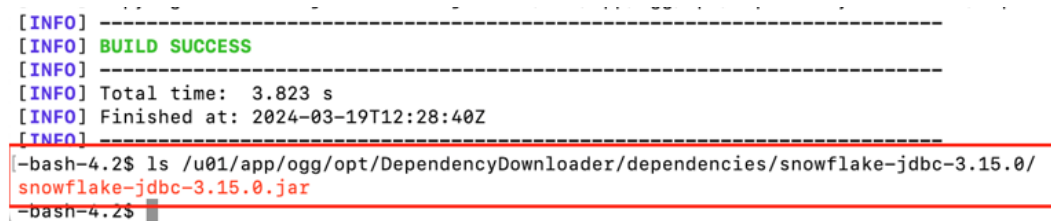
1. In your GoldenGate for VM, go to dependency downloader utility. It is located at `GG_HOME/opt/DependencyDownloader/` and locate `snowflake.sh`.
2. Run `snowflake.sh` with the required version. You can check the version and reported vulnerabilities in <https://mvnrepository.com/artifact/net.snowflake/snowflake-jdbc>. This document uses 3.15.0 which is the latest version when this quickstart is published.

Figure 8-1 Snowflake Ingestion - Install dependency



3. A new directory gets created in `GG_HOME/opt/DependencyDownloader/dependencies` named as `<snowflake_version>`. Make a note of this directory as it will be used in the `replicat` properties. For example: `/u01/app/ogg/opt/DependencyDownloader/dependencies/snowflake_3.15.0`

Figure 8-2 Snowflake jar new directory



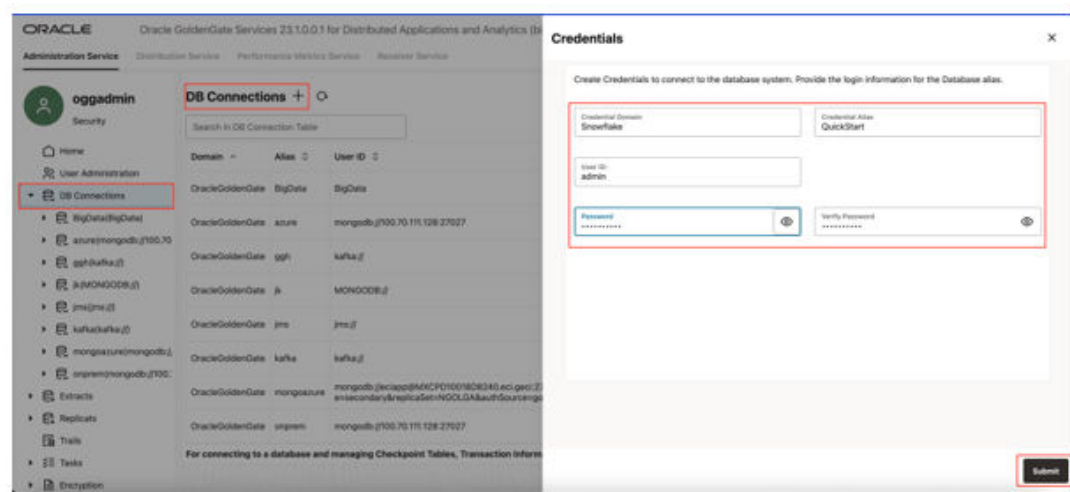
8.2.3 Create a Credential Store Entry

To create a credential store entry for securing Snowflake username and password:

1. In **Administration Service**, click **DB Connections** and then click **+** sign for adding new DB Connection.
2. In the **Credentials** window. Provide the following details:
 - **Credential Domain**: Domain name of your choice
 - **Credential Alias**: Alias name of your choice
 - **User ID**: Snowflake Username
 - **Password**: Snowflake Password

- **Verify Password:** Snowflake Password

Figure 8-3 Create a credential entry

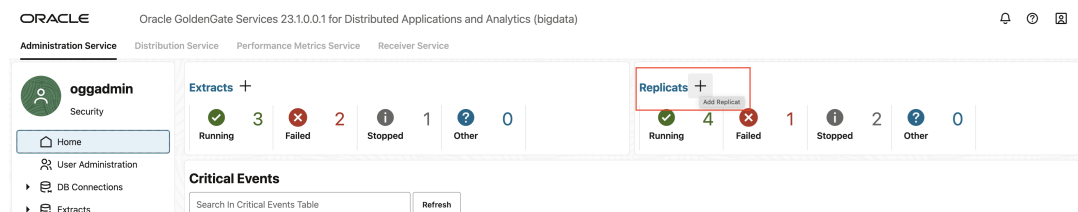


8.2.4 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

To create a replicat in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA):

1. In the Oracle GoldenGate for Distributed Applications and Analytics UI, in the **Administration Service** tab, click the **+** sign to add a replicat.

Figure 8-4 Click + in the Administration Service tab.



2. In the **Add Replicat** window, select the **Classic Replicat**.

Note

There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

3. Click **Next**. There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-5 Add a Replicat

Add Replicat

Replicat Information Replicat Options Managed Options Parameter File Properties File

Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.

Replicat Type

☒ Classic Replicat ☐ Coordinated Replicat

Process Name: Snowf

Description:

Next ➤

4. Enter the basic information, and click **Next**:
 - a. **Trail Name**: Name of the required trail file (if using sample trail, provide as tr)
 - b. **Subdirectory**: Provide as GG_HOME/opt/AdapterExamples/trail/ if using the sample trail.
 - c. **Target**: Snowflake Data Warehouse

Figure 8-6 Add Replicat - Replicat Options

Add Replicat

Replicat Information **Replicat Options** Managed Options Parameter File Properties File

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: tr Subdirectory: AdapterExamples/trail/j Encryption Profile: LocalWallet

Begin Position in Trail: Sequence Number

Trail Position

Sequence Number: 0 RSA Offset: 0

Target:

Target: Snowflake Data Warehouse

Stage and Merge using external object storage: ☐

Back **Next**

5. Leave **Managed Options** as is and click **Next**.
6. Enter **Parameter File** details and click **Next**. In the Parameter File, you can specify source to target mapping. If you are using the sample trail file (tr), then enter as follows: MAP QASOURCE.*, TARGET <your_schema_name>.*;
If Coordinated Replicat is selected as the Replicat Type, then an additional parameter needs to be provided:

TARGETDB LIBFILE libggjava.so SET property=<ggbd-deployment_home>/etc/conf/ogg/your_replicat_name.properties

Figure 8-7 Enter parameter details

Add Replicat

Replicat Information Replicat Options Managed Options **Parameter File** Properties File

Use parameter file to configure replicat parameters and source to target mapping.

```
REPLICAT Snowf
MAP QASOURCE.*, TARGET <your_schema_name>.*;
```

Back Next

7. In the next screen, update the properties only tagged as T0 D0 and click **Create**.

Edit your Snowflake JDBC URL:

```
#TODO: Edit JDBC ConnectionUrl
gg.eventhandler.snowflake.connectionURL=jdbc:snowflake://
<account_name>.snowflakecomputing.com/?<connection_params>
```

Edit Snowflake username and password:

```
#TODO: Edit JDBC user name with alias and domain from step 2.
gg.eventhandler.snowflake.UserName=ORACLEWALLETUSERNAME [alias domain]
#TODO: Edit JDBC password with alias and domain from step 2.
gg.eventhandler.snowflake.Password=ORACLEWALLETPASSWORD [alias domain]
```

Provide path to dependency jar files that you downloaded in prerequisites:

```
#TODO: Edit to include the GCS Java SDK and BQ Java SDK.
gg.classpath=/path/to/Snowflake-dependencies/*
```

Figure 8-8 Properties File

Add Replicat

Replicat Information Replicat Options Managed Options Parameter File **Properties File**

Configure replicat properties marked as #TODO.

```
# Properties file for Replicat Snowf
# Snowflake Data Warehouse Template
# Configuration to load GoldenGate trail operation records
# into Snowflake Data warehouse by chaining
# File writer handler -> Snowflake Event handler (uses Snowflake internal stage).
# Note: Recommended to only edit the configuration marked as TODO
gg.target=snowflake

#The Snowflake Event Handler
#TODO: Edit JDBC ConnectionUrl
gg.eventhandler.snowflake.connectionURL=jdbc:snowflake://ggtest.snowflakecomputing.com/?
warehouse=COMPUTE_WH&db=OCIGLL&CLIENT_SESSION_KEEP_ALIVE=true
#TODO: Edit JDBC user name
gg.eventhandler.snowflake.UserName=ORACLEWALLETUSERNAME [QuickStart Snowflake]
#TODO: Edit JDBC password
gg.eventhandler.snowflake.Password=ORACLEWALLETPASSWORD [QuickStart Snowflake]
#TODO: Set the classpath to include Snowflake JDBC driver.
gg.classpath=/u01/app/ogg/opt/DependencyDownloader/dependencies/Snowflake/*
#TODO: Provide sufficient memory (at least 8GB).
jvm.bootoptions=-Xmx8g -Xms8g
```

Back Create **Create and Run**

See [Understanding the BigQuery Handler Configuration](#) for more replicat configuration details.

8. If replicat starts successfully, it will be in running state. You can go to action/details/statistics to see the replication statistics.

Figure 8-9 Replicat Statistics

ORACLE Oracle GoldenGate Services 23.1.0.0.1 for Distributed Applications and Analytics (bigdata)

Administration Service Distribution Service Performance Metrics Service Recovery Service

oggsadmin Security

Home User Administration DB Connections Extracts **Replicats** Trails Tasks

Replicats + Group Action

Running 4 Failed 0 Stopped 2 Other 0

Search in Replicats Table

Process Name	Status	Lag	Actions
Snowf	Running	11 days 20 hours 43 minutes 0 seconds	ⓘ ⚙ ⌵ ⋮
JIT	Running	0 sec	ⓘ ⚙ ⌵ ⋮

Figure 8-10 Replicat Table Statistics

Oracle GoldenGate Services 23.10.0.1 for Distributed Applications and Analytics (bigdata)

Administration Service | Distribution Service | Performance Metrics Service | Replicat Service

Home | User Administration | DB Connections | Extracts | Replicats | JRT | Checkpoint | **Statistics** | Parameters | Properties | Log | Report

SNOWF

Statistics

Total | Daily | Hourly

Table Statistics

Search in Database Statistics Table

Table Name	Target Table	Inserts	Updates	Upserts	Deletes	Truncates	Ignores	Discards	Conflicts
GASOURCE.FOUSTMER	GASOURCE.FOUSTMER	5	1	0	0	0	0	0	0
GASOURCE.FOUSTORD	GASOURCE.FOUSTORD	5	3	0	2	0	0	0	0

- You can go to Snowflake console and check the tables. It may take a short while for tables to be created and loaded.

Note

- You can run an initial load with Snowflake replicat. For more details, see [Using OCI GoldenGate for Snowflake Initial Load and Real-time Data Sync](#)
- For all Snowflake handler configuration details, see [Snowflake](#).

8.3 Realtime Parquet Ingestion into Google Cloud Storage with GoldenGate for Distributed Applications and Analytics 23.8 and later

This Quickstart covers a step-by-step process showing how to ingest parquet files into Google Cloud Storage buckets in real-time with GoldenGate for Distributed Applications and Analytics (GG for DAA).

This Quickstart is valid for GG for DAA 23.7 and later versions.

Google Cloud Storage (GCS) is a service for storing objects in Google Cloud Platform.

GG for DAA GCS handler works in conjunction with File Writer Handler and Parquet Handler (if parquet is required). File Writer Handler produces files locally, optionally Parquet Handler converts to parquet format and GCS Handler loads into GCS buckets.

- [Prerequisites](#)
- [Install Required Dependency Files](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.3.1 Prerequisites

To successfully complete this quickstart, you must have the following:

- Google Cloud Storage Bucket
- [Google Service Account Key](#) with Bucket and Object Permissions. For more information, see [Replicate Data](#)

- Public access to your bucket (GG for DAA supports private bucket access, you can refer to GG for DAA document for details)

In this Quickstart, a sample trail file, such as `tr` which is shipped with GG for DAA is used. If you want to continue with sample trail file, then it is located at `GG_HOME/opt/AdapterExamples/trail/` in your GG for DAA instance.

8.3.2 Install Required Dependency Files

GG for DAA uses client libraries in the replication process and these libraries need to be downloaded before setting up the replication process. You can use [dependency downloader](#) to download the client libraries. Dependency downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

GG for DAA uses a 3-step process to ingest parquet into GCS buckets:

- Generating local files from trail files
- Converting local files to Parquet format
- Loading files into GCS

For generating local parquet files with GG for DAA, replicat uses [File Writer Handler](#) and [Parquet Handler](#). To load the parquet files into GCS, GG for DAA uses [Google Cloud Storage Handler](#) in conjunction with File Writer and Parquet Event Handler.

- In your GG for DAA VM, go to dependency downloader utility. It is located at `GG_HOME/opt/DependencyDownloader/`.
- Run `parquet.sh`, `hadoop.sh`, and `gcs.sh` with the required versions. You can check the version and reported vulnerabilities in Maven Central.

Figure 8-11 Install Required Dependency Files

```

-bash-4.2$ pwd
/u01/app/ogg/opt/DependencyDownloader
-bash-4.2$ ls
aws.sh          docs            hadoop_cloudera.sh  internal_scripts    kafka_mapr.sh      project
bigquery.sh     elasticsearch_rest.sh  hadoop_hortonworks.sh  kafka.sh            mongodb.sh          snowflake.sh
cassandra.sh    elasticsearch_transport.sh  hadoop_mapr.sh        kafka_cloudera.sh   oracle_nosql_sdk.sh  synapse.sh
cassandra_dse.sh  gcs.sh          hbase.sh            kafka_confluent.sh  oracle_oci.sh       velocity.sh
config_proxy.sh  hadoop.sh       hbase_cloudera.sh   kafka_confluent_protobuf.sh  orc.sh
dependencies     hadoop_azure_cloudera.sh  hbase_hortonworks.sh  kafka_hortonworks.sh  parquet.sh
-bash-4.2$ ./parquet.sh 1.12.2

labkeys — ssh -i id_rsa opc@141.144.231.135 — 150x33
-bash-4.2$ ls
aws.sh          docs            hadoop_cloudera.sh  internal_scripts    kafka_mapr.sh      project
bigquery.sh     elasticsearch_rest.sh  hadoop_hortonworks.sh  kafka.sh            mongodb.sh          snowflake.sh
cassandra.sh    elasticsearch_transport.sh  hadoop_mapr.sh        kafka_cloudera.sh   oracle_nosql_sdk.sh  synapse.sh
cassandra_dse.sh  gcs.sh          hbase.sh            kafka_confluent.sh  oracle_oci.sh       velocity.sh
config_proxy.sh  hadoop.sh       hbase_cloudera.sh   kafka_confluent_protobuf.sh  orc.sh
dependencies     hadoop_azure_cloudera.sh  hbase_hortonworks.sh  kafka_hortonworks.sh  parquet.sh
-bash-4.2$ ./hadoop.sh 3.1.1

-bash-4.2$ ls
abs          aeh3.xml      bq.xml        cassandra_capture_dse.sh  dependencies  hadoop.sh          hbase.sh          kafka_cloudera.sh  mongodb.sh
abs.xml      asb2.xml      bq.xml        cassandra_dse.sh         docs          hadoop_azure_cloudera.sh  hbase_cloudera.sh  kafka_confluent.sh  mongodb_capture.sh
aeh          aeh2.xml      cassandra.sh  config_proxy.sh          download_dependencies.sh  hadoop_cloudera.sh  hbase_hortonworks.sh  kafka_confluent_protobuf.sh  oracle_nosql_sdk.sh
aeh.xml     aws.sh        cassandra_capture_3x.sh  delta                  elasticsearch_java.sh    hadoop_hortonworks.sh  internal_scripts    kafka_hortonworks.sh  oracle_oci.sh
aeh3        bigquery.sh   cassandra_capture_4x.sh  delta.xml              gcs.sh          hadoop_mapr.sh      kafka.sh           kafka_mapr.sh       orc.sh
-bash-4.2$ ./gcs.sh 2.29.1

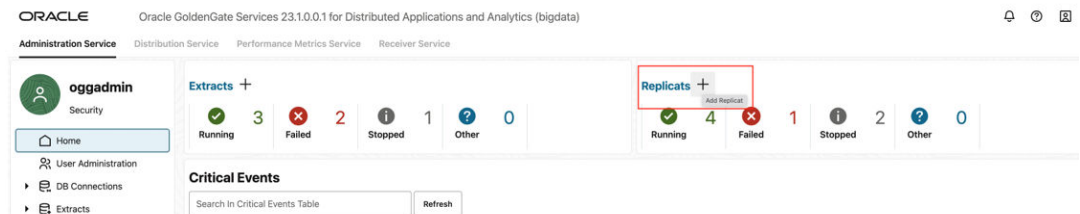
```

- 3 directories are created in `GG_HOME/opt/DependencyDownloader/dependencies`. Make a note of the directory. For example:
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/gcs_12.29.1`
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/hadoop_3.3.0`
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/parquet_1.12.3`

8.3.3 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

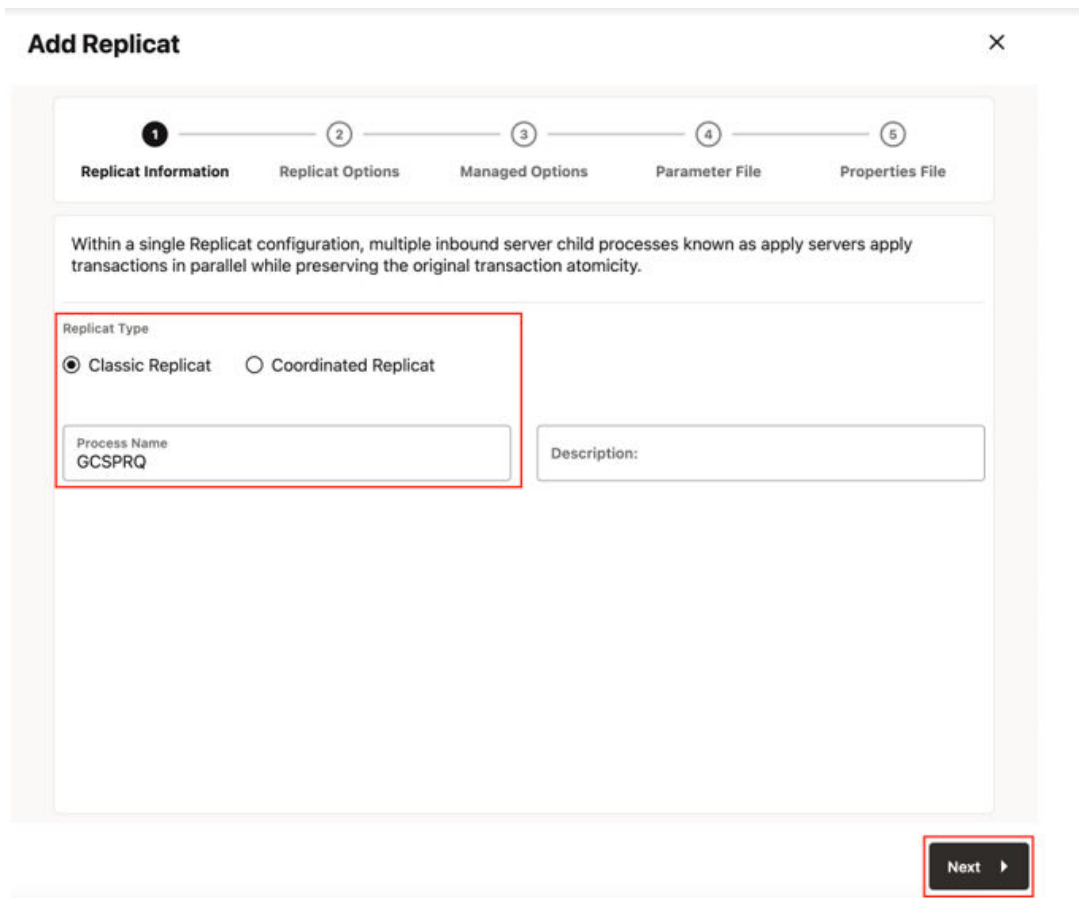
1. Go to **Administration Service** and click + sign to add a replicat.

Figure 8-12 Administration Service



2. Select the **Replicat Type**, enter the Process Name and click **Next**. There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-13 Replicat Type



3. Enter **Replicat Options** and click **Next**.

- **Replicat Trail:** Name of the required trail file. For sample trail, enter tr.
- **Subdirectory:** Enter as `GG_HOME/opt/AdapterExamples/trail/` if using the sample trail.
- **Target:** Google Cloud Storage
- **Format:** Select the file format

Figure 8-14 Replicat Options

Add Replicat [X]

Replicat Information [✓] Replicat Options [✓] Managed Options [✓] Parameter File [✓] Properties File [5]

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: tr Subdirectory: /u01/ogg/opt/AdapterE Encryption Profile: LocalWallet

Begin Position in Trail

Trail Position

Sequence Number: 0 RBA Offset: 0

Target Target: Google Cloud Storage

Format Format: Parquet

[Back] [Next]

4. Leave **Managed Options** as is and click **Next**.

Figure 8-15 Managed Options

Add Replicat

Replicat Information Replicat Options **Managed Options** Parameter File Properties File

Use managed options to manage replicat start and auto-start options.

Profile Name: **Default** Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes: 0 Startup Delay Seconds: 0

Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries: 9 Retry Delay Minutes: 0 Retry Delay Seconds: 0

Retries Window Hours: 0 Retries Window Minutes: 0 Retries Window Seconds: 0

◀ Back **Next ▶**

5. Provide **Parameter File** details and click **Next**. In the Parameter File, you can either specify source to target mapping or leave it as-is with a wildcard selection. If Coordinated Replicat is selected as the Replicat Type, then you need to provide an additional parameter as follows:
 TARGETDB LIBFILE libggjava.so SET property=<ggbd-deployment_home>/etc/conf/ogg/your_replicat_name.properties

Figure 8-16 Parameter File

Add Replicat

Replicat Information Replicat Options Managed Options **Parameter File** Properties File

Use parameter file to configure replicat parameters and source to target mapping.

```
REPLICAT GCSPRQ
MAP *.* TARGET *.*;
```

Right Drawer Popup

Back Next

6. In **Properties File**, update the properties marked as #TODO and click **Create and Run**. For example:

```
# Properties file for Replicat
gg.target=gcs
```

```
#TODO: format can be 'parquet' or 'orc' or one of the pluggable formatter types.
Default is 'parquet'.
gg.format=parquet
```

```
#The GCS Event handler
gg.eventhandler.gcs.pathMappingTemplate=${fullyQualifiedTableName}
#TODO: Edit the GCS bucket name
gg.eventhandler.gcs.bucketMappingTemplate=<gcs-bucket-name>
#TODO: Edit the GCS credentialsFile
gg.eventhandler.gcs.credentialsFile=/path/to/gcp/credentialsFile
gg.eventhandler.gcs.finalizeAction=none
```

```
#TODO: Edit to include the GCS Java SDK.
gg.classpath=/path/to/gcs-deps/:/path/to/hadoop-deps/:/path/to/parquet_deps/*
```

7. If replicat starts successfully, then it is in running state. You can go to replicat/statistics to see the replication statistics.

Figure 8-17 Replication Statistics

The screenshot displays the Oracle GoldenGate Services 23.10.0.1 for Distributed Applications and Analytics (bigdata) interface. The left sidebar shows the 'Administration Service' menu with options like Home, User Administration, DB Connections, Extracts, Replicats, Trails, and Tasks. The main panel shows the 'Replicats' section with a summary of replication status: 4 Running, 0 Failed, 2 Stopped, and 0 Other. Below this is a table of replicats with columns for Process Name, Status, Lag, and Actions. The table lists two replicats: 'GCSPRQ' with a status of 'Running' and a lag of '11 days 20 hours 43 minutes 6 seconds', and 'J11' with a status of 'Running' and a lag of '0 sec'. The bottom section shows the 'GCSPRQ' statistics, including a 'Table Statistics' table with columns for Table Name, Target Table, Inserts, Updates, Upserts, Deletes, Truncates, Ignores, Discards, and Conflicts. The table lists two tables: 'QASOURCE.TCUSTMER' and 'QASOURCE.TCUSTORD', both with a target table of 'QASOURCE.TCUSTMER' and 'QASOURCE.TCUSTORD' respectively. The statistics for 'QASOURCE.TCUSTMER' are: Inserts: 5, Updates: 1, Upserts: 0, Deletes: 0, Truncates: 0, Ignores: 0, Discards: 0, Conflicts: 0. The statistics for 'QASOURCE.TCUSTORD' are: Inserts: 5, Updates: 3, Upserts: 0, Deletes: 2, Truncates: 0, Ignores: 0, Discards: 0, Conflicts: 0.

8. You can go to GCP Cloud Storage bucket and check the files.

Figure 8-18 Google Cloud Storage Bucket

The screenshot shows the Google Cloud Storage bucket 'ggs_stage'. The bucket is publicly accessible because all users or all authenticated users have one or more permissions. The bucket is located in 'us-west2 (Los Angeles)' with a 'Standard' storage class and 'Public to internet' public access. The bucket is empty, showing a 'Folder browser' view with a search bar and a table of objects. The table has columns for Name, Size, Type, Created, Storage class, Last modified, Public access, Version history, Encryption, and Object retention. The only object listed is 'QASOURCE.TCUSTORD/'.

Note

- If target GCS bucket does not exist, it will be auto created by GG for DAA. You can use [Template Keywords](#) to dynamically assign the container names.
- GCS Event Handler can be configured for proxy server. For more information, see [Google Cloud Storage Event Handler](#).

8.4 Realtime Parquet Ingestion into Google Cloud Storage with Oracle GoldenGate for Distributed Applications and Analytics

Overview

This Quickstart covers a step-by-step process showing how to ingest parquet files into Google Cloud Storage buckets in real-time with GoldenGate for Distributed Applications and Analytics (GG for DAA).

Google Cloud Storage (GCS) is a service for storing objects in Google Cloud Platform. GG for DAA GCS handler works in conjunction with [File Writer Handler](#) and [Parquet Handler](#) (if parquet is required). File Writer Handler produces files locally, optionally Parquet Handler converts to parquet format and [GCS Handler](#) loads into GCS buckets.

- [Prerequisites](#)
- [Install Dependency Files](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.4.1 Prerequisites

To successfully complete this quickstart, you must have the following:

- Google Cloud Storage Bucket
- [Google Service Account Key](#) with [Bucket and Object Permissions](#)
- Public access to your bucket (GG for DAA supports private bucket access). For more information, see [Google Cloud Storage](#).

In this Quickstart, a sample trail file (named *tr*) which is shipped with GG for DAA is being used. The sample trail file, it is located at GG_HOME/opt/AdapterExamples/trail/ in your GG for DAA instance.

8.4.2 Install Dependency Files

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) uses client libraries in the replication process and these libraries need to be downloaded before setting up the replication process. You can use dependency downloader to download the client libraries. [Dependency Downloader](#) is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

GG for DAA uses a 3-step process to ingest parquet into GCS buckets:

- Generating local files from trail files
- Converting local files to Parquet format
- Loading files into GCS

For generating local parquet files with GG for DAA, replicat uses [File Writer Handler](#) and [Parquet Handler](#). To load the parquet files into GCS, GG for DAA uses [Google Cloud Storage Handler](#) in conjunction with File Writer and Parquet Event Handler.

To install the required dependency files:

1. In your GG for DAA VM, go to dependency downloader utility. It is located at GG_HOME/opt/DependencyDownloader/.

- Run `parquet.sh`, `hadoop.sh`, and `gcs.sh` with the required versions. You can check the version and reported vulnerabilities in Maven Central.

Figure 8-19 3 directories created in `GG_HOME/opt/DependencyDownloader/dependencies`

```

-bash-4.2$ pwd
/u01/app/ogg/opt/DependencyDownloader
-bash-4.2$ ls
aws.sh          docs            hadoop_cloudera.sh  internal_scripts    kafka_mapr.sh      project
bigquery.sh     elasticsearch_rest.sh  hadoop_hortonworks.sh  kafka.sh            mongodb.sh          snowflake.sh
cassandra.sh    elasticsearch_transport.sh  hadoop_mapr.sh        kafka_cloudera.sh   oracle_nosql_sdk.sh  synapse.sh
cassandra_dse.sh  gcs.sh          hbase.sh            kafka_confluent.sh  oracle_oci.sh       velocity.sh
config_proxy.sh  hadoop.sh       hbase_cloudera.sh   kafka_confluent_protobuf.sh  orc.sh              parquet.sh
dependencies     hadoop_azure_cloudera.sh  hbase_hortonworks.sh  kafka_hortonworks.sh
-bash-4.2$ ./parquet.sh 1.12.2

-bash-4.2$ ls
aws.sh          docs            hadoop_cloudera.sh  internal_scripts    kafka_mapr.sh      project
bigquery.sh     elasticsearch_rest.sh  hadoop_hortonworks.sh  kafka.sh            mongodb.sh          snowflake.sh
cassandra.sh    elasticsearch_transport.sh  hadoop_mapr.sh        kafka_cloudera.sh   oracle_nosql_sdk.sh  synapse.sh
cassandra_dse.sh  gcs.sh          hbase.sh            kafka_confluent.sh  oracle_oci.sh       velocity.sh
config_proxy.sh  hadoop_azure_cloudera.sh  hbase_hortonworks.sh  kafka_hortonworks.sh
dependencies     hadoop_3.1.1

-bash-4.2$ ls
aws.sh          aws.xml          bq.xml          cassandra_capture_dir.sh  dependencies  hadoop.sh          hbase.sh          kafka_cloudera.sh  mongodb.sh
db2.xml          db2.xml          bq.xml          cassandra_capture_dir.sh  download_dependencies.sh  hadoop_azure_cloudera.sh  hbase_cloudera.sh  kafka_confluent_protobuf.sh  mongodb_capture.sh
dell.xml         dell.xml         bq.xml          cassandra_capture_dir.sh  elasticsearch_java.sh    hadoop_cloudera.sh      hbase_hortonworks.sh  kafka_confluent_protobuf.sh  oracle_nosql_sdk.sh
dell.xml         dell.xml         bq.xml          cassandra_capture_dir.sh  gcs.sh            hadoop_hortonworks.sh  internal_scripts    kafka_mapr.sh          oracle_oci.sh
dell.xml         dell.xml         bq.xml          cassandra_capture_dir.sh  gcs.sh            hadoop_mapr.sh        kafka_hortonworks.sh  kafka_mapr.sh          orc.sh
-bash-4.2$ ./gcs.sh 12.29.1

```

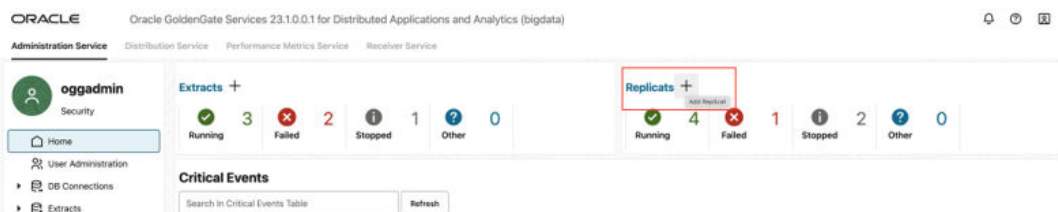
- Three new directories get created in `GG_HOME/opt/DependencyDownloader/dependencies` named as `<dependencyname_version>`. Make a note of these directories as they get used in the replicat properties. For example: `/u01/app/ogg/opt/DependencyDownloader/dependencies/gcs_12.29.1`, and `/u01/app/ogg/opt/DependencyDownloader/dependencies/hadoop_3.4.0`

8.4.3 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

To create a replicat in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA):

- In the GG for DAA UI, in the **Administration Service** tab, click the **+** sign to add a replicat.

Figure 8-20 Click the **Administration Service** tab



- Select the **Classic Replicat** Replicat Type and click **Next**. There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single-threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-21 Add Replicat

Add Replicat ×

1 — 2 — 3 — 4 — 5
Replicat Information Replicat Options Managed Options Parameter File Properties File

Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.

Replicat Type

☒ Classic Replicat ☐ Coordinated Replicat

Process Name
GCSPRQ

Description:

Next ►

3. Enter the Replicat information, and click **Next**:
 - a. **Replicat Trail**: Name of the required trail file. For sample trail, provide tr.
 - b. **Subdirectory**: Provide as GG_HOME/opt/AdapterExamples/trail/ if using the sample trail.
 - c. **Target**: Google Cloud Storage

Figure 8-22 Add Replicat

Add Replicat ×

1 ✓ 2 3 4 5
Replicat Information Replicat Options Managed Options Parameter File Properties File

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: tr Subdirectory: /AdapterExamples/trail/ Encryption Profile: LocalWallet

Begin Position in Trail: 0

Trail Position

Sequence Number: 0 RBA Offset: 0

Target:

Target: Google Cloud Storage

Back Next

4. Leave **Managed Options** as is and click **Next**.

Figure 8-23 Add Replicat - Managed Options

The screenshot shows the 'Add Replicat' wizard with five steps: Replicat Information, Replicat Options, Managed Options (current), Parameter File, and Properties File. The 'Managed Options' step is active, showing a progress bar with steps 1, 2, 3, 4, and 5. Below the progress bar, the text reads: 'Use managed options to manage replicat start and auto-start options.'

The form contains the following fields and controls:

- Profile Name:** A dropdown menu with 'Default' selected.
- Critical to deployment health:** A toggle switch currently turned off.
- Auto Start:** A toggle switch currently turned off.
- Startup Delay Minutes:** A text input field with '0'.
- Startup Delay Seconds:** A text input field with '0'.
- Auto Restart:** A toggle switch currently turned off.
- Restart on Failure only:** A toggle switch currently turned off.
- Disable Task After Retries:** A toggle switch currently turned off.
- Max Retries:** A text input field with '9' and up/down arrows.
- Retry Delay Minutes:** A text input field with '0' and a 'Right Drawer Popup' tooltip.
- Retry Delay Seconds:** A text input field with '0'.
- Retries Window Hours:** A text input field with '0'.
- Retries Window Minutes:** A text input field with '0'.
- Retries Window Seconds:** A text input field with '0'.

At the bottom right, there are two buttons: 'Back' and 'Next'. The 'Next' button is highlighted with a red rectangle.

5. Enter **Parameter File** details and click **Next**. In the Parameter File, you can either specify source to target mapping or leave it as-is with a wildcard selection. If Coordinated Replicat is selected as the Replicat Type, then an additional parameter needs to be provided:
 TARGETDB LIBFILE libggjava.so SET property=<ggbd-deployment_home>/etc/conf/ogg/your_replicat_name.properties

Figure 8-24 Add Replicat - Enter Parameter Details

Add Replicat ✕

✓ ✓ ✓ 4 5
 Replicat Information Replicat Options Managed Options **Parameter File** Properties File

Use parameter file to configure replicat parameters and source to target mapping.

```

REPLICAT GCSPRQ
MAP ***, TARGET ***;
    
```

Right Drawer Popup

◀ Back
Next ▶

6. In the Properties File, remove all the pre-configured properties; but not the first row marked with the replicat name (# Properties file for Replicat <replicat_name>). Copy and paste below property list into properties file, update the properties marked as #TODO and then click **Create and Run**.

#The File Writer Handler - no need to change

```

gg.handlerlist=filewriter
gg.handler.filewriter.type=filewriter
gg.handler.filewriter.mode=op
gg.handler.filewriter.pathMappingTemplate=./dirout
gg.handler.filewriter.stateFileDirectory=./dirsta
gg.handler.filewriter.fileRollInterval=7m
gg.handler.filewriter.inactivityRollInterval=5s
gg.handler.filewriter.fileWriteActiveSuffix=.tmp
gg.handler.filewriter.finalizeAction=delete
    
```

Avro OCF - no need to change

```

gg.handler.filewriter.format=avro_row_ocf
gg.handler.filewriter.fileNameMappingTemplate=${groupName}_${
fullyQualifiedTableName}_${currentTimestamp}.avro
gg.handler.filewriter.format.pkUpdateHandling=delete-insert
gg.handler.filewriter.format.metaColumnsTemplate=${optype},${position}
gg.handler.filewriter.format.iso8601Format=false
gg.handler.filewriter.partitionByTable=true
gg.handler.filewriter.rollOnShutdown=true
    
```

#The Parquet Event Handler - no need to change

```

gg.handler.filewriter.eventHandler=parquet
gg.eventhandler.parquet.type=parquet
gg.eventhandler.parquet.pathMappingTemplate=./dirparquet
gg.eventhandler.parquet.fileNameMappingTemplate=${groupName}_${fullyQualifiedTableName}_${currentTimestamp}.parquet
gg.eventhandler.parquet.writeToHDFS=false
gg.eventhandler.parquet.finalizeAction=delete

```

#Select GCS Event Handler - no need to change

```

gg.eventhandler.parquet.eventHandler=gcs

```

#TODO Set GCS Event handler - please update as needed

```

gg.eventhandler.gcs.type=gcs
gg.eventhandler.gcs.pathMappingTemplate=${fullyQualifiedTableName}
#TODO: Edit the GCS bucket name
gg.eventhandler.gcs.bucketMappingTemplate=<bucket_name>
#TODO: Edit the GCS credentialsFile
gg.eventhandler.gcs.credentialsFile=path_to_GCP_Credential_File
gg.eventhandler.gcs.finalizeAction=none

```

#TODO Set the classpath with the paths you noted in step1

```

gg.classpath=path_to/ gcs_12.29.1/: path_to /hadoop_3.4.0/: path_to/
parquet_1.12.3/* jvm.bootoptions=-Xmx512m -Xms32m

```

7. If replicat starts successfully, then it is in running state. You can go to Replicats/Statistics to see the replication statistics.

Figure 8-25 Replicats Statistics

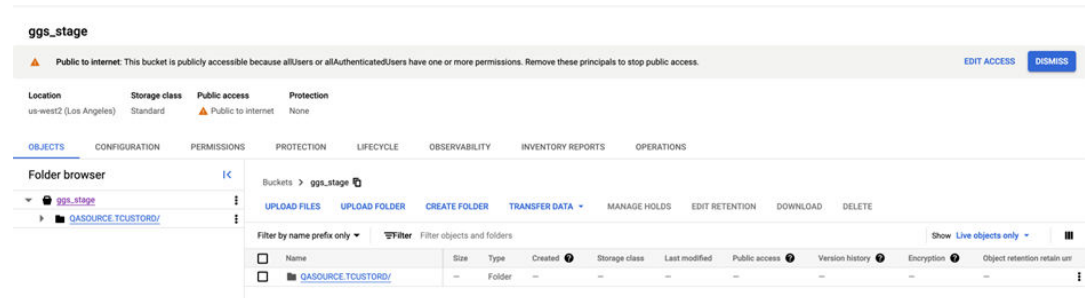
The screenshot displays the Oracle GoldenGate Services 23.1.0.0.1 for Distributed Applications and Analytics (bigdata) interface. The top section shows 'Replicats' with a summary of 4 Running, 0 Failed, 2 Stopped, and 0 Other. Below this is a table of replicats. The bottom section shows 'Statistics' for the selected replicat 'GCSPRQ', including a 'Table Statistics' table.

Process Name	Status	Lag	Actions
GCSPRQ	Running	11 days 20 hours 43 minutes 6 seconds	[Info] [Refresh] [Delete] [More]
J/J1	Running	0 sec	[Info] [Refresh] [Delete] [More]

The 'Statistics' section for 'GCSPRQ' includes a 'Table Statistics' table.

Table Name	Target Table	Inserts	Updates	Upserts	Deletes	Truncates	Ignores	Discards	Conflicts
QASOURCE.TCUSTMER	QASOURCE.TCUSTMER	5	1	0	0	0	0	0	[Info] [Refresh] [Delete] [More]
QASOURCE.TCUSTORD	QASOURCE.TCUSTORD	5	3	0	2	0	0	0	[Info] [Refresh] [Delete] [More]

8. Go to GCP Cloud Storage bucket and check the files.

Figure 8-26 GCP Cloud Storage Bucket**Note**

- If target GCS bucket does not exist, it will be auto created by GG for DAA. You can use [Template Keywords](#) to dynamically assign the container names.
- GCS Event Handler can be configured for proxy server. For more information, see [Replicate Data](#).
- You can use different properties to control the behaviour of file writing. You can set file sizes, inactivity periods, and more. You can get more details in the File Writer [blog post](#).

8.5 Realtime Parquet Ingestion into AWS S3 Buckets with Oracle GoldenGate for Distributed Applications and Analytics 23.8 and later

This Quickstart covers a step-by-step process showing how to ingest parquet files into AWS S3 buckets in real-time with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

Amazon Simple Storage Service (Amazon S3) is an object storage service provided by Amazon Web Services.

GG for DAA S3 handler works in conjunction with File Writer Handler and Parquet Handler (if parquet is required). File Writer Handler produces files locally, optionally Parquet Handler converts to parquet format and S3 Handler loads into S3 buckets.

- [Prerequisites](#)
- [Install Required Dependency Files](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.5.1 Prerequisites

To successfully complete this Quickstart, you must have the following:

- Amazon S3 Bucket
- [Amazon S3 Access Key & Secret](#)

In this Quickstart, a sample trail file (named `tr`) which is shipped with GG for DAA is used. If you want to continue with sample trail file, then it is located at `GG_HOME/opt/AdapterExamples/trail/` in your GG for DAA instance.

8.5.2 Install Required Dependency Files

GG for DAA uses client libraries in the replication process and these libraries need to be downloaded before setting up the replication process. You can use [dependency downloader](#) to download the client libraries. Dependency downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

GG for DAA uses a 3-step process to ingest parquet into s3 buckets:

- Generating local files from trail files
- Converting local files to Parquet format
- Loading files into AWS S3 buckets

For generating local parquet files with GG for DAA, replicat uses [File Writer Handler](#) and [Parquet Handler](#). To load the parquet files into AWS S3, GG for DAA uses [S3 Event Handler](#) in conjunction with File Writer and Parquet Event Handler.

GG for DAA uses 3 different set of client libraries to create parquet files and loading into AWS S3.

1. In your GG for DAA VM, go to dependency downloader utility. It is located at `GG_HOME/opt/DependencyDownloader/`
2. Run `parquet.sh`, `hadoop.sh`, and `s3.sh` with the required versions.

Note

Running `s3.sh` downloads every dependency needed for AWS sdk version 2.x, if it is required to download the dependencies for version 1.x, use the `aws.sh` instead.

Figure 8-27 Install Required Dependency Files

```

:DependencyDownloader$ ls
aws.sh                cassandra_dse.sh      gcs.sh                hbase_hortonworks.sh  kafka.sh              orc.sh                snowflakestreaming.sh
azure_blob_storage.sh cassandra.sh           googlepubsub.sh       hbase.sh              kinesis.sh            parquet.sh            synapse.sh
bigquery.sh           config_proxy.sh        hadoop_azure_cloudera.sh internal_scripts       mongodb_capture.sh    project              velocity.sh
bigquerystreaming.sh databricks.sh          hadoop_cloudera.sh    kafka_cloudera.sh     mongodb.sh            redis.sh              xmls
cassandra_capture_3x.sh docs                  hadoop_hortonworks.sh kafka_confluent_protobuf.sh onelake.sh            s3.sh                snowflake-fips.sh
cassandra_capture_4x.sh download_dependencies.sh hadoop.sh              kafka_confluent.sh    kafka_confluent.sh    oracle_nosql_sdk.sh  snowflake-oci.sh    snowflake.sh
cassandra_capture_dse.sh elasticsearch_java.sh  hbase_cloudera.sh     kafka_hortonworks.sh  oracle_oci.sh         snowflake.sh

:DependencyDownloader$ ./parquet.sh 1.12.2

:DependencyDownloader$ ls
aws.sh                cassandra_dse.sh      gcs.sh                hbase_hortonworks.sh  kafka.sh              orc.sh                snowflakestreaming.sh
azure_blob_storage.sh cassandra.sh           googlepubsub.sh       hbase.sh              kinesis.sh            parquet.sh            synapse.sh
bigquery.sh           config_proxy.sh        hadoop_azure_cloudera.sh internal_scripts       mongodb_capture.sh    project              velocity.sh
bigquerystreaming.sh databricks.sh          hadoop_cloudera.sh    kafka_cloudera.sh     mongodb.sh            redis.sh              xmls
cassandra_capture_3x.sh docs                  hadoop_hortonworks.sh kafka_confluent_protobuf.sh onelake.sh            s3.sh                snowflake-fips.sh
cassandra_capture_4x.sh download_dependencies.sh hadoop.sh              kafka_confluent.sh    kafka_confluent.sh    oracle_nosql_sdk.sh  snowflake-oci.sh    snowflake.sh
cassandra_capture_dse.sh elasticsearch_java.sh  hbase_cloudera.sh     kafka_hortonworks.sh  oracle_oci.sh         snowflake.sh

:DependencyDownloader$ ./hadoop.sh 3.1.1

:DependencyDownloader$ ls
aws.sh                cassandra_dse.sh      gcs.sh                hbase_hortonworks.sh  kafka.sh              orc.sh                snowflakestreaming.sh
azure_blob_storage.sh cassandra.sh           googlepubsub.sh       hbase.sh              kinesis.sh            parquet.sh            synapse.sh
bigquery.sh           config_proxy.sh        hadoop_azure_cloudera.sh internal_scripts       mongodb_capture.sh    project              velocity.sh
bigquerystreaming.sh databricks.sh          hadoop_cloudera.sh    kafka_cloudera.sh     mongodb.sh            redis.sh              xmls
cassandra_capture_3x.sh docs                  hadoop_hortonworks.sh kafka_confluent_protobuf.sh onelake.sh            s3.sh                snowflake-fips.sh
cassandra_capture_4x.sh download_dependencies.sh hadoop.sh              kafka_confluent.sh    kafka_confluent.sh    oracle_nosql_sdk.sh  snowflake-oci.sh    snowflake.sh
cassandra_capture_dse.sh elasticsearch_java.sh  hbase_cloudera.sh     kafka_hortonworks.sh  oracle_oci.sh         snowflake.sh

:DependencyDownloader$ ./s3.sh 2.28.11
  
```

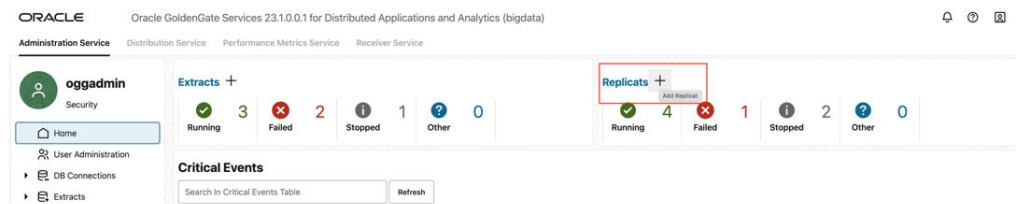
3. 3 directories will be created in `GG_HOME/opt/DependencyDownloader/dependencies`. Make a note of these directories:


```
/u01/app/ogg/opt/DependencyDownloader/dependencies/s3_2.28.11/*  
/u01/app/ogg/opt/DependencyDownloader/dependencies/hadoop_3.4.0/*  
/u01/app/ogg/opt/DependencyDownloader/dependencies/parquet_1.12.3/*
```

8.5.3 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

1. Go to **Administration Service** and click **+** sign to add a replicat.

Figure 8-28 Administration Service



2. Select the Replicat Type and click **Next**.
There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-29 Replicat Options

The screenshot shows the 'Add Replicat' dialog box with a progress bar at the top indicating five steps: 1. Replicat Information, 2. Replicat Options (current step), 3. Managed Options, 4. Parameter File, and 5. Properties File. Below the progress bar, a text box explains: 'Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.' The 'Replicat Type' section has two radio buttons: 'Classic Replicat' (selected) and 'Coordinated Replicat'. Below this, there are two text input fields: 'Process Name' with the value 'AWSS3' and 'Description' with the value 'AWS S3 Replication'. At the bottom right, there is a 'Next' button with a right-pointing arrow. A red box highlights the 'Replicat Type' section and the 'Next' button. A 'Right Drawer Popup' label is visible at the bottom left of the main content area.

3. Provide **Replicat Options** and click **Next**.

- **Replicat Trail:** Name of the required trail file. For sample trail, provide tr.
- **Subdirectory:** Provide as GG_HOME/opt/AdapterExamples/trail/ if using the sample trail.
- **Target:** Amazon S3
- **Format:** Select the file format.

Figure 8-30 Add Replicate

Add Replicate

Replicat Information Replicat Options Managed Options Parameter File Properties File

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: tr Subdirectory: /u01/ogg/opt/AdapterE Encryption Profile: LocalWallet

Begin Position in Trail: Begin

Trail Position

Sequence Number: 0 RBA Offset: 0

Target

Target: Amazon S3

Format

Format: Parquet

Back Next

4. Leave **Managed Options** as is and click **Next**.

Figure 8-31 Managed Options

Add Replicat

Replicat Information Replicat Options **Managed Options** Parameter File Properties File

Use managed options to manage replicat start and auto-start options.

Profile Name: **Default** Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes: 0 Startup Delay Seconds: 0

Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries: 9 Retry Delay Minutes: 0 Retry Delay Seconds: 0

Retries Window Hours: 0 Retries Window Minutes: 0 Retries Window Seconds: 0

Back **Next**

5. In **Parameter File**, provide parameter file details and click **Next**.
In the Parameter File, you can either specify source to target mapping or leave it as-is with a wildcard selection. If Coordinated Replicat is selected as the Replicat Type, an additional parameter needs to be provided:

```
TARGETDB LIBFILE libggjava.so SET property=<ggbd-deployment_home>/etc/conf/ogg/
your_replicat_name.properties
```

Figure 8-32 Parameter File

Add Replicat

Replicat Information Replicat Options Managed Options **Parameter File** Properties File

Use parameter file to configure replicat parameters and source to target mapping.

```
REPLICAT AWSS3
MAP *.* , TARGET *.*;
```

Back Next

6. In **Properties File**, update the properties marked as #TODO and click **Create and Run**.
For example:

Properties file for Replicat

S3 gg.target=s3

#TODO: format can be 'parquet' or 'orc' or one of the pluggable formatter types.
Default is 'parquet'.

gg.format=parquet

#TODO: Update S3 region

gg.eventhandler.s3.region=

#TODO: Uncomment and configure the proxy host and port.

#gg.eventhandler.s3.proxyServer=

#gg.eventhandler.s3.proxyPort=

#TODO: Update S3 bucket details

gg.eventhandler.s3.bucketMappingTemplate=

gg.eventhandler.s3.pathMappingTemplate=

gg.eventhandler.s3.finalizeAction=none

#TODO: Set the AWS credentials.

gg.eventhandler.s3.accessKeyId=

gg.eventhandler.s3.secretKey=

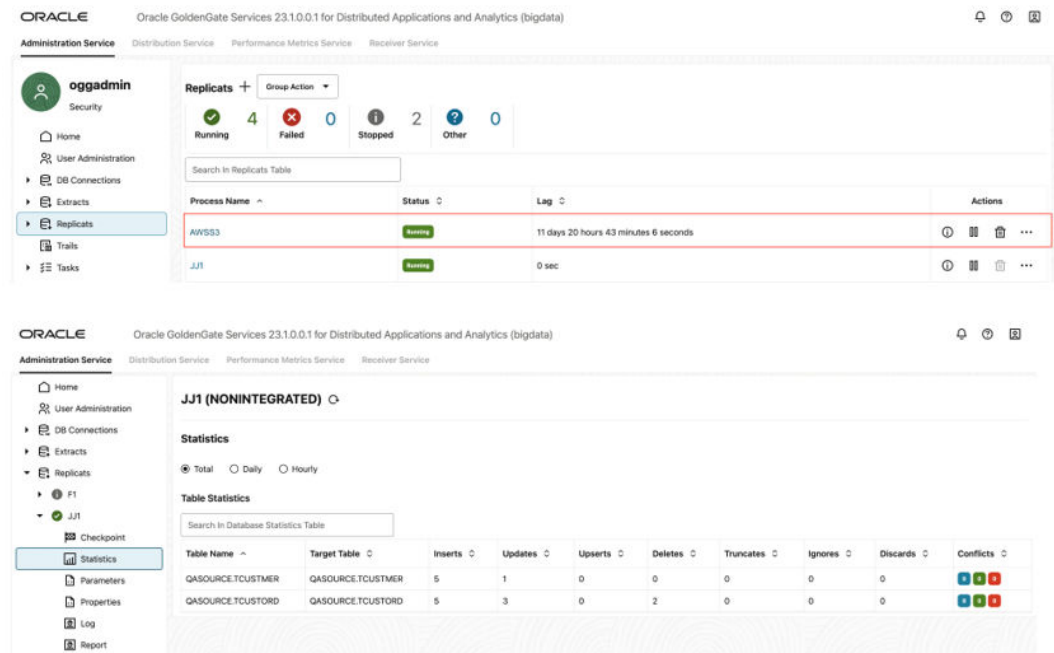
gg.eventhandler.s3.enableBucketAdmin=false

#TODO: Set to the location of the AWS SDK.

gg.classpath=/path/to/aws-deps/*:/path/to/hadoop-deps/*:/path/to/parquet_deps/*

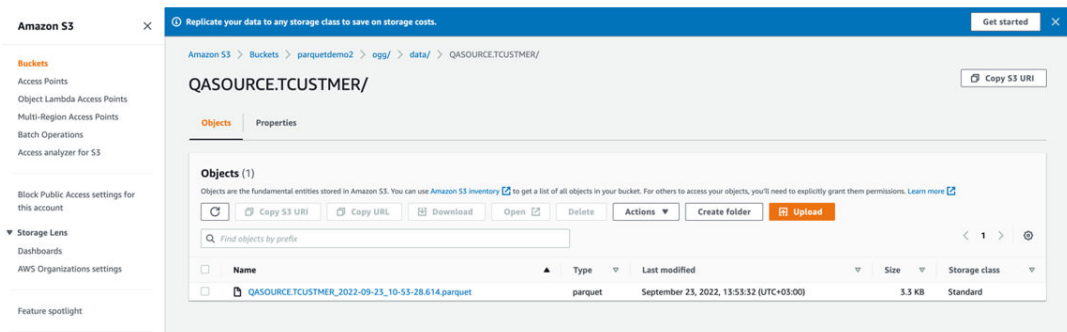
7. When replicat starts successfully, it is in running state. You can click on the replicat name, select **Statistics** to see the replication statistics.

Figure 8-33 Replication Statistics



8. You can go to AWS s3 console and check the bucket.

Figure 8-34 AWS s3 console.



Note

- If target S3 bucket does not exist, it can be auto created by GGBD. You can use [Template Keywords](#) to dynamically assign S3 bucket names.
- S3 Handler can be configured for proxy server. For more information, see [S3 Event Handler](#).
- You can use different properties to control the behaviour of file writing. You can set file sizes, inactivity periods and more. You can get more details in the [File Writer blog post](#).

8.6 Realtime Parquet Ingestion into AWS S3 Buckets with Oracle GoldenGate for Distributed Applications and Analytics

Overview

This Quickstart covers a step-by-step process showing how to ingest parquet files into AWS S3 buckets in real-time with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

Amazon Simple Storage Service (Amazon S3) is an object storage service provided by Amazon Web Services.

GG for DAA S3 handler works in conjunction with [File Writer Handler](#) and [Parquet Handler](#) (if parquet is required). File Writer Handler produces files locally, optionally Parquet Handler converts to parquet format and [S3 Handler](#) loads into S3 buckets.

- [Prerequisites](#)
- [Install Dependency Files](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.6.1 Prerequisites

To successfully complete this Quickstart, you must have the following:

- Amazon S3 Bucket
- [Amazon S3 Access Key & Secret](#)

In this Quickstart, a sample trail file (named *tr*) which is shipped with GG for DAA is used. If you want to continue with sample trail file, then it is at GG_HOME/opt/AdapterExamples/trail/ in your GG for DAA instance.

8.6.2 Install Dependency Files

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) uses client libraries in the replication process and these libraries need to be downloaded before setting up the replication process. You can use dependency downloader to download the client libraries. [Dependency Downloader](#) is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

GG for DAA uses a 3-step process to ingest parquet into s3 buckets:

- Generating local files from trail files
- Converting local files to Parquet format
- Loading files into AWS s3 buckets

For generating local parquet files with GG for DAA, replicat uses [File Writer Handler](#) and [Parquet Event Handler](#). To load the parquet files into AWS s3, GG for DAA uses [S3 Event Handler](#) in conjunction with File Writer and Parquet Event Handler.

GG for DAA uses 3 different set of client libraries to create parquet files and loading into AWS s3:

1. In your GG for DAA VM, go to Dependency Downloader utility. It is located at:
GG_HOME/opt/DependencyDownloader/

2. Run `parquet.sh` and `hadoop.sh` with and `aws.sh` with the required versions.

Figure 8-35 II. Run `parquet.sh` and `hadoop.sh` with and `aws.sh` with the required versions

The figure consists of three terminal screenshots showing a directory listing and the execution of scripts. The directory listing shows various subdirectories including `aws.sh`, `docs`, `elasticsearch_rest.sh`, `elasticsearch_transport.sh`, `gcs.sh`, `hadoop.sh`, `hadoop_azure_cloudera.sh`, `dependencies`, `hadoop_cloudera.sh`, `hadoop_hortonworks.sh`, `hbase.sh`, `hbase_cloudera.sh`, `hbase_hortonworks.sh`, `internal_scripts`, `kafka.sh`, `kafka_cloudera.sh`, `kafka_confluent.sh`, `kafka_confluent_protobuf.sh`, `kafka_hortonworks.sh`, `kafka_mapr.sh`, `mongodb.sh`, `oracle_nosql_sdk.sh`, `oracle_oci.sh`, `orc.sh`, `parquet.sh`, `project`, `snowflake.sh`, `synapse.sh`, `velocity.sh`, and `velocity.sh`. The first screenshot shows the execution of `parquet.sh 1.12.2`. The second screenshot shows the execution of `hadoop.sh 3.1.1`. The third screenshot shows the execution of `aws.sh 1.12.309`.

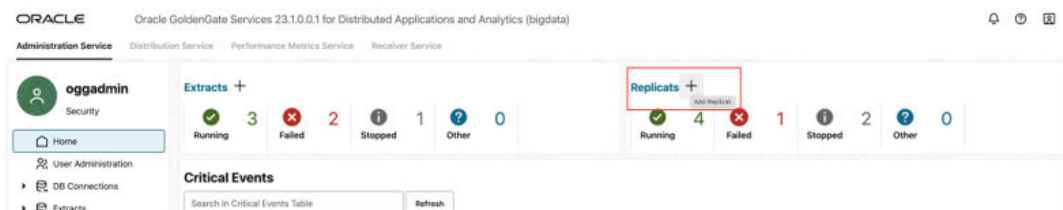
3. 3 directories are created in `GG_HOME/opt/DependencyDownloader/dependencies`. Note the directories. For example:
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/aws_sdk_1.12.309/*`
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/hadoop_3.4.0/*`
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/parquet_1.12.3/*`

8.6.3 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

To create a replicat in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA):

1. In the Oracle GoldenGate for Distributed Applications and Analytics UI, in the **Administration Service** tab, click the **+** sign to add a replicat.

Figure 8-36 Click the Administration Service tab



2. Select the **Classic Replicat** Replicat Type and click **Next**. There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-37 Add Replicat

Add Replicat

1 Replicat Information 2 Replicat Options 3 Managed Options 4 Parameter File 5 Properties File

Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.

Replicat Type

☒ Classic Replicat ☐ Coordinated Replicat

Process Name: AWSS3

Description: AWS S3 Replication

Next

3. Enter the Replicat information, and click **Next**:
 - a. **Replicat Trail**: Name of the required trail file. For sample trail, enter tr.
 - b. **Target**: Amazon s3

Figure 8-38 Enter Replicat Details

Add Replicat

Replicat Information Replicat Options Managed Options Parameter File Properties File

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: tr Subdirectory: Encryption Profile: LocalWallet

Begin Position in Trail: 0

Trail Position

Sequence Number: 0 RBA Offset: 0

Target:

Target: Amazon S3

Back Next

4. Leave **Managed Options** as is and click **Next**.

Figure 8-39 Add Replicat - Managed Options

Add Replicat

Replicat Information Replicat Options **Managed Options** Parameter File Properties File

Use managed options to manage replicat start and auto-start options.

Profile Name: **Default** Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes: 0 Startup Delay Seconds: 0

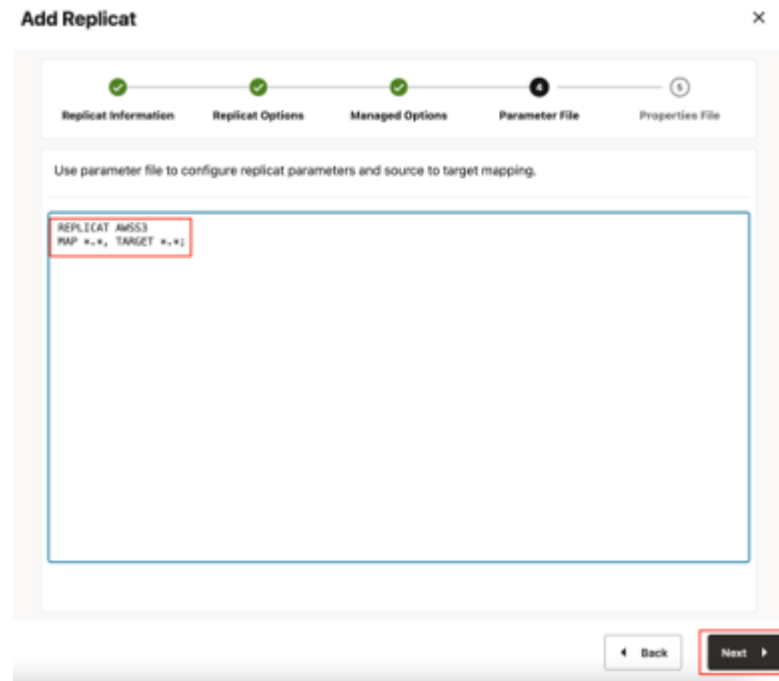
Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries: 9 Retry Delay Minutes: 0 Retry Delay Seconds: 0

Retries Window Hours: 0 Retries Window Minutes: 0 Retries Window Seconds: 0

Back **Next**

5. Enter **Parameter File** details and click **Next**. In the Parameter File, you can either specify source to target mapping or leave it as-is with a wildcard selection. If **Coordinated Replicat** is selected as the Replicat Type, then an additional parameter needs to be provided: TARGETDB LIBFILE libggjava.so SET property=<ggbd-deployment_home>/etc/conf/ogg/your_replicat_name.properties

Figure 8-40 Parameter File

6. In the Properties File, remove all the pre-configured properties; but not the first row marked with the replicat name (# Properties file for Replicat <replicat_name>). Copy and paste the following property list into the properties file, update the properties marked as #TODO and click **Create and Run**.

#The File Writer Handler - no need to change

```
gg.handlerlist=filewriter
gg.handler.filewriter.type=filewriter
gg.handler.filewriter.mode=op
gg.handler.filewriter.pathMappingTemplate=./dirout
gg.handler.filewriter.stateFileDirectory=./dirsta
gg.handler.filewriter.fileRollInterval=7m
gg.handler.filewriter.inactivityRollInterval=5s
gg.handler.filewriter.fileWriteActiveSuffix=.tmp
gg.handler.filewriter.finalizeAction=delete
```

Avro OCF - no need to change

```
gg.handler.filewriter.format=avro_row_ocf
gg.handler.filewriter.fileNameMappingTemplate=${groupName}_${fullyQualifiedTableName}_${currentTimestamp}.avro
gg.handler.filewriter.format.pkUpdateHandling=delete-insert
gg.handler.filewriter.format.metaColumnsTemplate=${optype},${position}
gg.handler.filewriter.format.iso8601Format=false
gg.handler.filewriter.partitionByTable=true
gg.handler.filewriter.rollOnShutdown=true
```

#The Parquet Event Handler - no need to change

```
gg.handler.filewriter.eventHandler=parquet
gg.eventhandler.parquet.type=parquet
gg.eventhandler.parquet.pathMappingTemplate=./dirparquet
gg.eventhandler.parquet.fileNameMappingTemplate=${groupName}_${fullyQualifiedTableName}_${currentTimestamp}.parquet
gg.eventhandler.parquet.writeToHDFS=false
gg.eventhandler.parquet.finalizeAction=delete
```

#TODO Select S3 Event Handler - no need to change

```
gg.eventhandler.parquet.eventHandler=s3
```

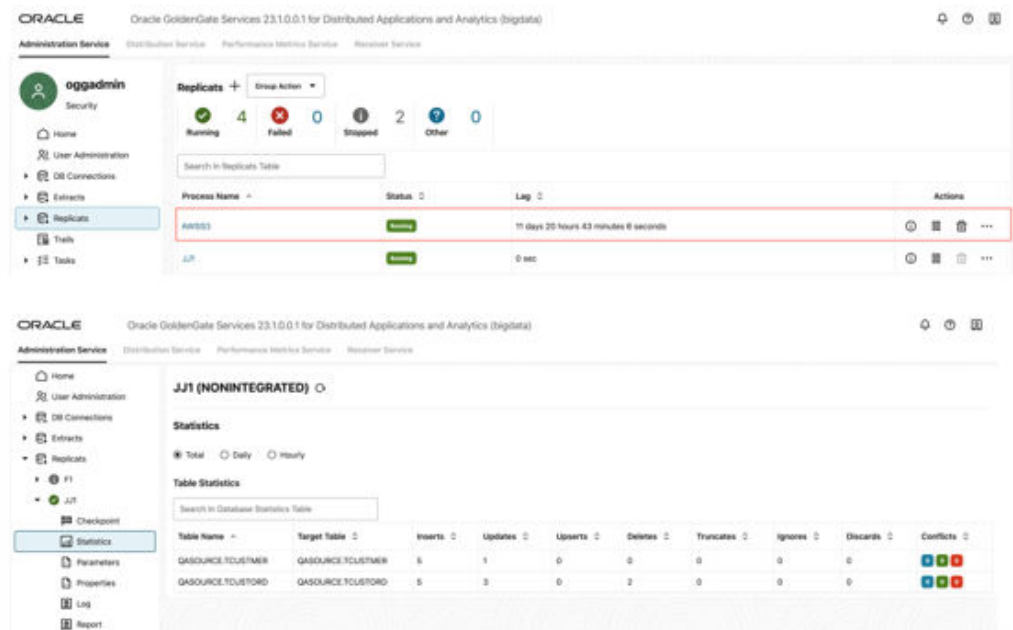
#TODO Set S3 Event Handler- please update as needed

```
gg.eventhandler.s3.type=s3
gg.eventhandler.s3.region=<your-aws-region>
gg.eventhandler.s3.bucketMappingTemplate=<target_s3_bucket_name>
gg.eventhandler.s3.pathMappingTemplate=ogg/data/${fullyQualifiedTableName}
gg.eventhandler.s3.accessKeyId=<provide_key>
gg.eventhandler.s3.secretKey=<provide_secret>
```

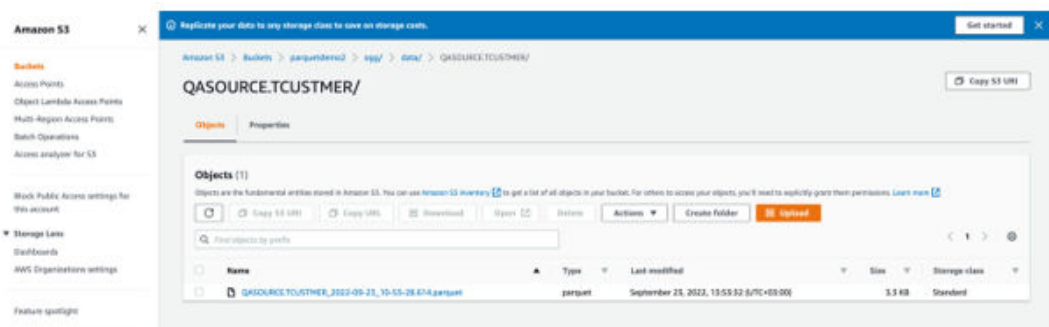
#TODO Set the classpath to the paths you noted in step1

```
gg.classpath=path_to/ gcs_12.29.1/: path_to /hadoop_3.4.0/: path_to/parquet_1.12.3/*
jvm.bootoptions=-Xmx512m -Xms32m
```

7. If replicat starts successfully, then it is in running state. You can go to Replicats/Statistics to see the replication statistics.

Figure 8-41 Replication Statistics

8. Go to AWS s3 console and check the files.

Figure 8-42 AWS S3 console

① Note

- If target s3 does not exist, then it will be auto created by GG for DAA. You can use [Template Keywords](#) to dynamically assign the container names.
- s3 Handler can be configured for proxy server. For more information, see [S3 Event Handler](#).
- You can use different properties to control the behaviour of file writing. You can set file sizes, inactivity periods and more. You can get more details in the File Writer [blog post](#).

8.7 Realtime Parquet Ingestion into Azure Data Lake Storage with Oracle GoldenGate for Distributed Applications and Analytics 23.8 and later

Overview

This Quickstart covers a step-by-step process showing how to ingest parquet files into Azure Storage containers in real-time with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

Azure Data Lake Storage (ADLS) is a centralized repository provided by Azure where you can store all your data, both structured and unstructured.

GG for DAA ADLS handler works in conjunction with File Writer Handler and Parquet Handler (if parquet is required). File Writer Handler produces files locally. Optionally, Parquet Handler converts to parquet format and ADLS Handler loads into Azure Storage containers.

GG for DAA provides different alternatives for ADLS connection. This Quickstart uses BLOB endpoint for connection.

- [Prerequisites](#)
- [Install Required Dependency Files](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

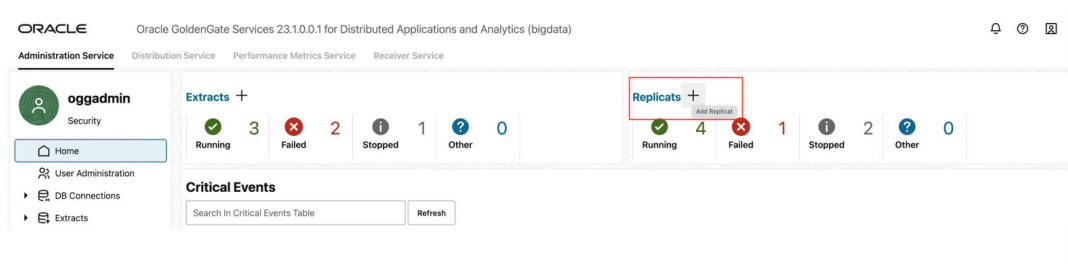
8.7.1 Prerequisites

To successfully complete this Quickstart, you must have the following:

- Azure Storage Account and a Container
- Azure Storage Account login credentials

In this Quickstart, a sample trail file, such as `tr`, which is shipped with GG for DAA is used. If you want to continue with sample trail file, then it is located at `GG_HOME/opt/AdapterExamples/trail` in your GG for DAA instance.

Figure 8-44 Administration Service



2. Select the **Replicat Type**, provide **Process Name** and click **Next**. There are two different Replicat types available: **Classic** and **Coordinated**. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-45 Add Replicat

Add Replicat ×

1 — 2 — 3 — 4 — 5

Replicat Information Replicat Options Managed Options Parameter File Properties File

Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.

Replicat Type

☒ Classic Replicat ☐ Coordinated Replicat

Process Name
ADLSRQ
Enter 8 or fewer characters.

Description:

Next ▶

3. Enter **Replicat Options** and click **Next**.
 - **Replicat Trail:** Name of the required trail file. For sample trail, provide tr.
 - **Subdirectory:** Provide as GG_HOME/opt/AdapterExamples/trail/ if using the sample trail.
 - **Target:** Azure Data Lake Storage

- **Format:** Select the format

Figure 8-46 Replicat Options

Add Replicat [X]

Replicat Information Replicat Options Managed Options Parameter File Properties File

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: tr Subdirectory: /u01/ogg/opt/AdapterE Encryption Profile: LocalWallet

Begin Position in Trail

Trail Position

Sequence Number: 0 RBA Offset: 0

Target Target: Azure Data Lake Storage

Format Format: Parquet

[Back] [Next]

4. Leave **Managed Options** as is and click **Next**.

Figure 8-47 Managed Options

Add Replicat [X]

Replicat Information [✓] Replicat Options [✓] **Managed Options** [3] Parameter File [4] Properties File [5]

Use managed options to manage replicat start and auto-start options.

Profile Name: **Default** Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes: 0 Startup Delay Seconds: 0

Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries: 9 Retry Delay Minutes: 0 Right Drawer Popup Retry Delay Seconds: 0

Retries Window Hours: 0 Retries Window Minutes: 0 Retries Window Seconds: 0

◀ Back **Next ▶**

5. Provide **Parameter File** details and click **Next**.

In the **Parameter File**, you can either specify source to target mapping or leave it as-is with a wildcard selection. If Coordinated Replicat is selected as the Replicat Type, an additional parameter needs to be provided: TARGETDB LIBFILE libggjava.so SET property=<ggbdd-deployment_home>/etc/conf/ogg/your_replicat_name.properties.

Figure 8-48 Parameter File

Add Replicat ✕

✓ Replicat Information
✓ Replicat Options
✓ Managed Options
4 **Parameter File**
5 Properties File

Use parameter file to configure replicat parameters and source to target mapping.

```
REPLICAT ADLSPRO
MAP *.* , TARGET *.*;
```

◀ Back
Next ▶

6. In the properties file, update the properties marked as #TODO and click **Create and Run**.

Properties file for Replicat

```
# Configuration to load GoldenGate trail operation records
# into Azure Data Lake Storage by chaining
# File writer handler -> Azure Blob Event handler.
# Note: Recommended to only edit the configuration marked as TODO
```

```
gg.target=abs
```

```
#TODO: format can be 'parquet' or 'orc' or one of the pluggable formatter types.
Default is 'parquet'.
gg.format=parquet
```

```
#TODO: Edit the Azure Blob Storage container name
gg.eventhandler.abs.bucketMappingTemplate=<abs-container-name>
#TODO: Edit the Azure storage account name.
gg.eventhandler.abs.accountName=<storage-account-name>
#TODO: Edit the Azure storage account key.
#gg.eventhandler.abs.accountKey=<storage-account-key>
#TODO: Edit the Azure shared access signature(SAS) to authenticate to an Azure
Service.
#gg.eventhandler.abs.sasToken=<sas-token>
#TODO: Edit the the tenant ID of the application.
gg.eventhandler.abs.tenantId=<azure-tenant-id>
#TODO: Edit the the client ID of the application.
```

```
gg.eventhandler.abs.clientId=<azure-client-id>
#TODO: Edit the the client secret for the authentication.
gg.eventhandler.abs.clientSecret=<azure-client-secret>
gg.classpath=/path/to/abs-deps/*:/path/to/hadoop-deps/*:/path/to/parquet_deps/*
#TODO: Edit the proxy configuration.
#jvm.bootoptions=-Dhttps.proxyHost=some-proxy-address.com -Dhttps.proxyPort=80 -
Djava.net.useSystemProxies=true
```

7. If replicat starts successfully, it will be in running state. You can go to replicat/statistics to see the replication statistics.

Figure 8-49 Replication Statistics

The screenshot shows the Oracle GoldenGate Services 23.10.0.1 for Distributed Applications and Analytics (bigdata) interface. The left sidebar shows the navigation menu with 'Replicats' selected. The main panel displays the 'Replicats' table with columns: Process Name, Status, Lag, and Actions. The table shows two replicats: ADLSPRQ (Running) and JJ1 (Running). The ADLSPRQ replicat is highlighted with a red box. Below the table, the 'ADLSPRQ' statistics are shown, including a 'Table Statistics' table with columns: Table Name, Target Table, Inserts, Updates, Upserts, Deletes, Truncates, Ignores, Discards, and Conflicts. The ADLSPRQ replicat is highlighted with a red box.

Process Name	Status	Lag	Actions
ADLSPRQ	Running	11 days 20 hours 43 minutes 6 seconds	[Icons]
JJ1	Running	0 sec	[Icons]

Table Name	Target Table	Inserts	Updates	Upserts	Deletes	Truncates	Ignores	Discards	Conflicts
QASOURCE.TCUSTMER	QASOURCE.TCUSTMER	5	1	0	0	0	0	0	0
QASOURCE.TCUSTORD	QASOURCE.TCUSTORD	5	3	0	2	0	0	0	0

8. You can go to Azure Storage console and check the container.

Figure 8-50 Azure Storage console

The screenshot shows the Azure Storage console interface for the 'dtest' container. The left sidebar shows the navigation menu with 'Containers' selected. The main panel displays the 'dtest' container details, including the authentication method (Microsoft Entra user account) and the location (dtest / ogg / QASOURCE.TCUSTMER). The 'Table Statistics' table is shown, listing the contents of the container. The table has columns: Name, Modified, Access tier, Archive status, Blob type, Size, and Lease state. The table shows one blob: QASOURCE.TCUSTMER_2024_03_15.parquet, which was modified on 4/2/2024 at 4:39:51 PM, has an access tier of 'Hot (Inferred)', is a 'Block blob', and has a size of 3.32 KiB. The blob is highlighted with a red box.

Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
QASOURCE.TCUSTMER_2024_03_15.parquet	4/2/2024 4:39:51 PM	Hot (Inferred)		Block blob	3.32 KiB	Available

Note

- If target Azure container does not exist, then it will be auto crated by GG for DAA. You can use [Template Keywords](#) to dynamically assign the container names.
- ABS Event Handler can be configured for proxy server. For more information, see [Replicate Data](#).
- You can use different properties to control the behaviour of file writing. You can set file sizes, inactivity periods and more. For more information, see the File Writer [blog post](#).

8.8 Realtime Parquet Ingestion into Azure Data Lake Storage with Oracle GoldenGate for Distributed Applications and Analytics

Overview

This Quickstart covers a step-by-step process showing how to ingest parquet files into Azure Storage containers in real-time with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

Azure Data Lake Storage (ADLS) is a centralized repository provided by Azure where you can store all your data, both structured and unstructured.

GG for DAA ADLS handler works in conjunction with [File Writer Handler](#) and [Parquet Handler](#) (if parquet is required). File Writer Handler produces files locally, optionally, Parquet Handler converts to parquet format and [Azure Data Lake Storage \(ADLS\) Handler](#) loads into Azure Storage containers.

GG for DAA provides different alternatives for ADLS connection. This Quickstart uses BLOB endpoint for connection.

- [Prerequisites](#)
- [Install Dependency Files](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.8.1 Prerequisites

To successfully complete this Quickstart, you must have the following:

- Azure Storage Account and a Container
- Azure Storage Account login credentials

In this Quickstart, a sample trail file (named *tr*), which is shipped with GG for DAA is used. The sample trail file is located at `GG_HOME/opt/AdapterExamples/trail/` in your GG for DAA instance.

8.8.2 Install Dependency Files

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) uses Java SDK provided by Azure. You can download the SDKs using [Dependency Downloader](#) utility shipped

with GG for DAA. Dependency downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

GG for DAA uses a 3-step process to ingest parquet into Azure Storage containers:

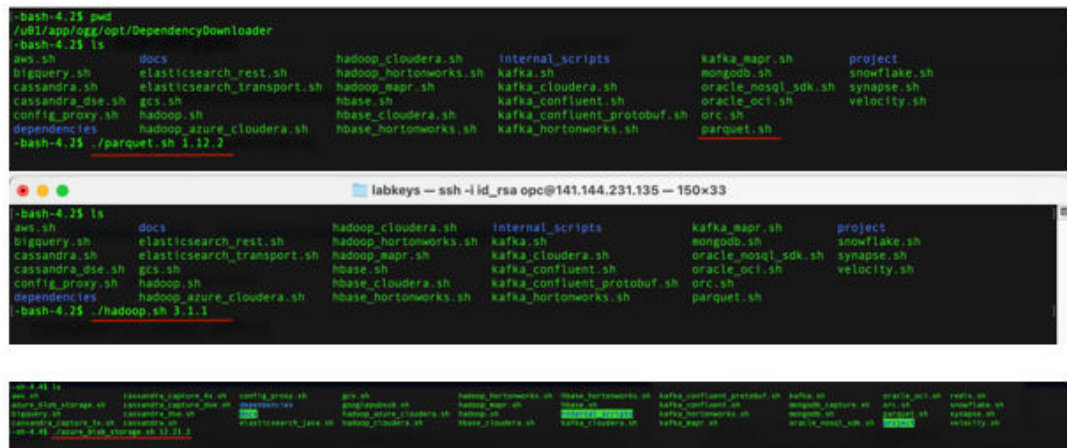
- Generating local files from trail files
- Converting local files to Parquet format
- Loading files into into Azure Storage containers

For generating local parquet files with GG for DAA, replicat uses [File Writer Handler](#) and [Parquet Handler](#). To load the parquet files into Azure Storage, GG for DAA uses [Azure BLOB Handler](#) in conjunction with File Writer and Parquet Event Handler.

GG for DAA uses 3 different set of client libraries to create parquet files and loading into Azure Storage:

1. In your GG for DAA VM, go to Dependency Downloader utility. It is located at `GG_HOME/opt/DependencyDownloader/`
2. Run `parquet.sh` and `hadoop.sh` with and `azure_blob_storage.sh` with the required versions.

Figure 8-51 II. Run `parquet.sh` and `hadoop.sh` with and `azure_blob_storage.sh` with the required versions.

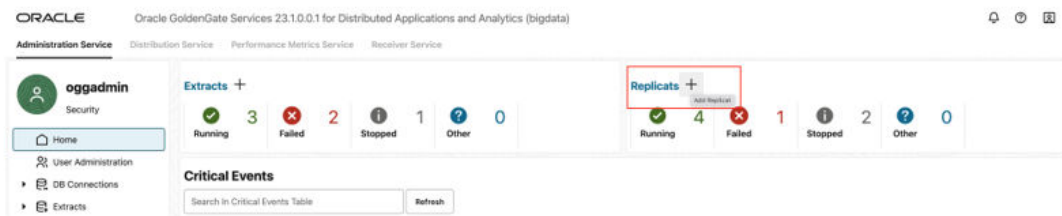


3. 3 directories are created in `GG_HOME/opt/DependencyDownloader/dependencies`. Note the directories. For example:
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/azure_blob_storage_12.21.2`
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/hadoop_3.4.0`
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/parquet_1.12.3`

8.8.3 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

To create a replicat in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA):

1. In the GG for DAA UI, in the **Administration Service** tab, click the + sign to add a replicat.

Figure 8-52 Click the Administration Service tab

2. Select the **Classic Replicat** Replicat Type and click **Next**. There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-53 Add Replicat

Add Replicat

1 Replicat Information 2 Replicat Options 3 Managed Options 4 Parameter File 5 Properties File

Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.

Replicat Type

☒ Classic Replicat ☐ Coordinated Replicat

Process Name
ADLSPRQ
Enter 8 or fewer characters.

Description:

Next

3. Enter the Replicat information, and click **Next**:
 - a. **Replicat Trail**: Name of the required trail file. For sample trail, provide tr.
 - b. **Subdirectory**: Enter GG_HOME/opt/AdapterExamples/trail/ if using the sample trail.
 - c. **Target**: Azure Data Lake Storage

Figure 8-54 Replicat Options

Add Replicat

Replicat Information **Replicat Options** Managed Options Parameter File Properties File

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: Subdirectory: Encryption Profile:

Begin:

Trail Position

Sequence Number: RBA Offset:

Target:

Target:

4. Leave **Managed Options** as is and click **Next**.

Figure 8-55 Managed Options

Add Replicat

Replicat Information Replicat Options **Managed Options** Parameter File Properties File

Use managed options to manage replicat start and auto-start options.

Profile Name: Default Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes 0 Startup Delay Seconds 0

Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries 9 Retry Delay Minutes 0 Retry Delay Seconds 0

Retries Window Hours 0 Retries Window Minutes 0 Retries Window Seconds 0

Back Next

5. Enter **Parameter File** details and click **Next**. In the Parameter File, you can either specify source to target mapping or leave it as-is with a wildcard selection. If Coordinated Replicat is selected as the Replicat Type, then an additional parameter needs to be provided: `TARGETDB LIBFILE libggjava.so SET property=<ggbd-deployment_home>/etc/conf/ogg/your_replicat_name.properties`

Figure 8-56 Parameter File

Add Replicat

Replicat Information Replicat Options Managed Options **Parameter File** Properties File

Use parameter file to configure replicat parameters and source to target mapping.

```
REPLICAT ADLSPRQ
MAP *.* TARGET *.*;
```

Back Next

6. In Properties File, remove all the pre-configured properties; but not the first row marked with the replicat name (# Properties file for Replicat <replicat_name>). Copy and paste below property list into properties file, update the properties marked as #TODO and click **Create and Run**.

#The File Writer Handler - no need to change

```
gg.handlerlist=filewriter
gg.handler.filewriter.type=filewriter
gg.handler.filewriter.mode=op
gg.handler.filewriter.pathMappingTemplate=./dirout
gg.handler.filewriter.stateFileDirectory=./dirsta
gg.handler.filewriter.fileRollInterval=7m
gg.handler.filewriter.inactivityRollInterval=5s
gg.handler.filewriter.fileWriteActiveSuffix=.tmp
gg.handler.filewriter.finalizeAction=delete
```

Avro OCF - no need to change

```
gg.handler.filewriter.format=avro_row_ocf
gg.handler.filewriter.fileNameMappingTemplate=${groupName}_${fullyQualifiedTableName}_${currentTimestamp}.avro
gg.handler.filewriter.format.pkUpdateHandling=delete-insert
gg.handler.filewriter.format.metaColumnsTemplate=${optype},${position}
gg.handler.filewriter.format.iso8601Format=false
gg.handler.filewriter.partitionByTable=true
gg.handler.filewriter.rollOnShutdown=true
```

#The Parquet Event Handler - no need to change

```

gg.handler.filewriter.eventHandler=parquet
gg.eventhandler.parquet.type=parquet
gg.eventhandler.parquet.pathMappingTemplate=./dirparquet
gg.eventhandler.parquet.fileNameMappingTemplate=${groupName}_${
fullyQualifiedTableName}_${currentTimestamp}.parquet
gg.eventhandler.parquet.writeToHDFS=false
gg.eventhandler.parquet.finalizeAction=delete

```

#TODO Select ABS Event Handler - no need to change

```
gg.eventhandler.parquet.eventHandler=abs
```

#TODO Set ABS Event Handler - please update as needed

```

gg.eventhandler.abs.type=abs
gg.eventhandler.abs.bucketMappingTemplate= <abs-container-name>
gg.eventhandler.abs.pathMappingTemplate=ogg/data/${fullyQualifiedTableName}
gg.eventhandler.abs.accountName= <storage-account-name>
#TODO: Edit the Azure storage account key if access key is used
#gg.eventhandler.abs.accountKey=<storage-account-key>
#TODO: Edit the Azure shared access signature (SAS) to if SAS is used.
#gg.eventhandler.abs.sasToken=<sas-token>
#TODO: Edit the the tenant ID, Client ID and Secret of the application if LDAP is
used.
#gg.eventhandler.abs.tenantId=<azure-tenant-id>
#gg.eventhandler.abs.clientId=<azure-client-id>
#gg.eventhandler.abs.clientSecret=<azure-client-secret>

```

#TODO Set the classpath to the paths you noted in step1

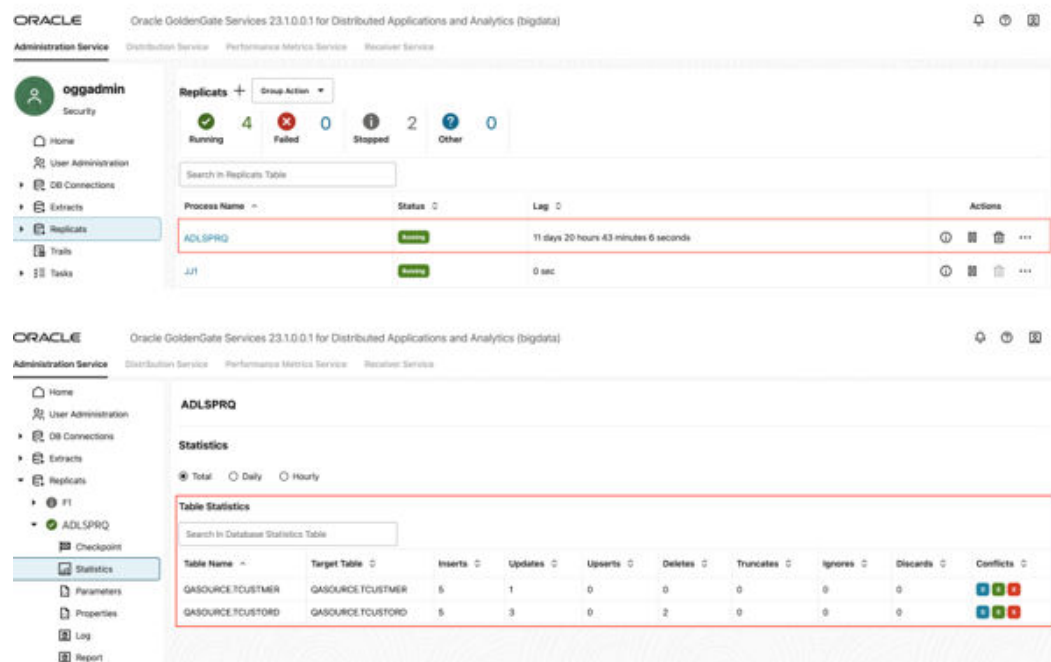
```

gg.classpath= path_to/ gcs_12.29.1/: path_to /hadoop_3.4.0/:path_to/parquet_1.12.3/*
jvm.bootoptions=-Xmx512m -Xms32m

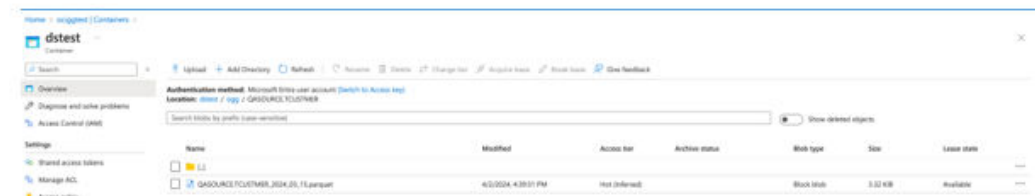
```

7. If replicat starts successfully, then it is in running state. You can go to Replicats/Statistics to see the replication statistics.

Figure 8-57 Replicats Statistics



8. Go to Azure Storage console and check the files.

Figure 8-58 Azure Storage console**Note**

- If target Azure container does not exist, it will be auto created by GG for DAA. You can use [Template Keywords](#) to dynamically assign the container names.
- ABS Event Handler can be configured for proxy server. For more information, see [Azure Blob Storage](#)
- You can use different properties to control the behaviour of file writing. You can set file sizes, inactivity periods and more. You can get more details in the File Writer [blog post](#).

8.9 Realtime Parquet Ingestion into OCI Object Storage with Oracle GoldenGate for Distributed Applications and Analytics 23.8 and later

This Quickstart covers a step-by-step process showing how to ingest parquet files into OCI Object Storage buckets in real-time with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

OCI Object Storage is a scalable, high-performance storage service provided by Oracle Cloud Infrastructure (OCI). It enables users to store and manage large amounts of unstructured data, such as images, videos, log files, backups, and other types of files.

GG for DAA OCI Object Storage handler works in conjunction with File Writer Handler and Parquet Handler (if parquet is required). File Writer Handler produces files locally, optionally Parquet Handler converts to parquet format and OCI Object Storage Handler loads into OCI Object Storage buckets.

- [Prerequisites](#)
- [Install Required Dependency Files](#)
- [Configure Credentials for Oracle Cloud Infrastructure](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.9.1 Prerequisites

To successfully complete this Quickstart, you must have the following:

- OCI Object Storage access

In this Quickstart, we will use a sample trail file (named tr) which is shipped with GG for DAA. If you want to continue with sample trail file, it is located at `GG_HOME/opt/AdapterExamples/trail/` in your GG for DAA instance.

8.9.2 Install Required Dependency Files

GG for DAA uses client libraries in the replication process and these libraries need to be downloaded before setting up the replication process. You can use [dependency downloader](#) to download the client libraries. Dependency downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

- GG for DAA uses a 3-step process to ingest parquet into OCI Object Storage buckets:
- Generating local files from trail files
- Converting local files to Parquet format
- Loading files into OCI Object Storage buckets

For generating local parquet files with GG for DAA, replicat uses [File Writer Handler](#) and [Parquet Handler](#). To load the parquet files into GG for DAA uses [OCI Event Handler](#) in conjunction with File Writer and Parquet Event Handler.

GG for DAA uses 3 different set of client libraries to create parquet files and loading into AWS S3.

1. In your GG for DAA VM, go to dependency downloader utility. It is located at `GG_HOME/opt/DependencyDownloader/`
2. Run `parquet.sh`, `hadoop.sh`, and `oracle_oci.sh` with the required versions.

Figure 8-59 Install required dependency files

```

-bash-4.2$ pwd
/u01/app/ogg/opt/DependencyDownloader
-bash-4.2$ ls
aws.sh          docs          hadoop_cloudera.sh  internal_scripts  kafka_mapr.sh    project
bigquery.sh     elasticsearch_rest.sh  hadoop_hortonworks.sh  kafka.sh          mongodb.sh       snowflake.sh
cassandra.sh    elasticsearch_transport.sh  hadoop_mapr.sh        kafka_cloudera.sh  oracle_nosql_sdk.sh  synapse.sh
cassandra_dse.sh  gcs.sh         hbase.sh            kafka_confluent.sh  oracle_oci.sh      velocity.sh
config_proxy.sh  hadoop.sh      hbase_cloudera.sh   kafka_confluent_protobuf.sh  orc.sh
dependencies     hadoop_azure_cloudera.sh  hbase_hortonworks.sh  kafka_hortonworks.sh  parquet.sh
-bash-4.2$ ./parquet.sh 1.12.2

labkeys — ssh -i id_rsa opc@141.144.231.135 — 150x33

-bash-4.2$ ls
aws.sh          docs          hadoop_cloudera.sh  internal_scripts  kafka_mapr.sh    project
bigquery.sh     elasticsearch_rest.sh  hadoop_hortonworks.sh  kafka.sh          mongodb.sh       snowflake.sh
cassandra.sh    elasticsearch_transport.sh  hadoop_mapr.sh        kafka_cloudera.sh  oracle_nosql_sdk.sh  synapse.sh
cassandra_dse.sh  gcs.sh         hbase.sh            kafka_confluent.sh  oracle_oci.sh      velocity.sh
config_proxy.sh  hadoop.sh      hbase_cloudera.sh   kafka_confluent_protobuf.sh  orc.sh
dependencies     hadoop_azure_cloudera.sh  hbase_hortonworks.sh  kafka_hortonworks.sh  parquet.sh
-bash-4.2$ ./hadoop.sh 3.1.1

-bash-4.2$ cd /u01/app/ogg/opt/DependencyDownloader/
-bash-4.2$ ls
abs          cassandra_capture_3k.sh  docs          download_dependencies.sh  hadoop_azure_cloudera.sh  hbase_hortonworks.sh  kafka_hortonworks.sh  kafka_mapr.sh  kafka_nosql_sdk.sh  mongo_4_6_0_dependencies.zip  oracle_oci.sh  project  snowflake.sh  synapse.sh  velocity.sh
abs2.xml     cassandra_dse.sh        elasticsearch_rest.sh  hadoop_hortonworks.sh  hadoop_mapr.sh           hbase.sh              kafka.sh              kafka_cloudera.sh  kafka_confluent.sh  kafka_confluent_protobuf.sh  kafka_hortonworks.sh  kafka_mapr.sh  kafka_nosql_sdk.sh  mongo_4_6_0_dependencies.zip  oracle_nosql_sdk.sh  orc.sh  parquet.sh  snowflake.sh  synapse.sh
aws.sh       config_proxy.sh          gcs.sh         hadoop.sh               hbase_cloudera.sh        hbase_hortonworks.sh  kafka_confluent.sh  kafka_confluent_protobuf.sh  kafka_hortonworks.sh  kafka_mapr.sh  kafka_nosql_sdk.sh  mongo_4_6_0_dependencies.zip  oracle_nosql_sdk.sh  orc.sh  parquet.sh  snowflake.sh  synapse.sh
bigquery.sh  dependencies             hadoop_azure_cloudera.sh  hbase_hortonworks.sh  kafka_hortonworks.sh     kafka_mapr.sh         kafka_nosql_sdk.sh  mongo_4_6_0_dependencies.zip  oracle_nosql_sdk.sh  orc.sh  parquet.sh  snowflake.sh  synapse.sh
-bash-4.2$ ./oracle_oci.sh 3.2.0

-bash-4.2$ cd dependencies/
-bash-4.2$ ls
aws.sh  hadoop_3.3.0  oracle_oci_3.2.0  parquet_1.12.3
-bash-4.2$

```

3. 3 directories are created in `GG_HOME/opt/DependencyDownloader/dependencies`. Make a note of these directories.
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/oracle_oci_3.2.0/*:`
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/hadoop_3.3.0/*:`
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/parquet_1.12.3/*`

8.9.3 Configure Credentials for Oracle Cloud Infrastructure

You need to create a configuration file to authenticate into OCI. The ideal configuration file include user, fingerprint, and key_file, tenancy, and region with their respective values. The default configuration file name and location is `~/.oci/config`. You can refer to [required keys and ocids](#) document for details.

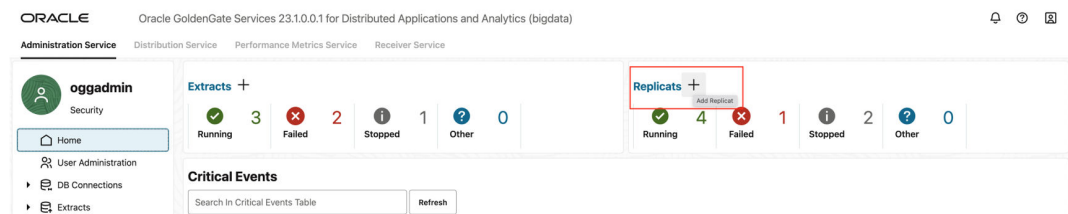
Sample config file

```
[DEFAULT]
user=<your_user_ocid>
fingerprint=<your_fingerprint>
key_file=~/.oci/oci_api_key.pem #path-to_your_key_file
tenancy=<your_tenancy_ocid>
```

8.9.4 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

1. Go to **Administration Service** and click + sign to add a replicat.

Figure 8-60 Administration Service



2. Select the **ReplicatType** and click **Next**
There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel. Coordinated Replicat results in multiple files being created as there is a multithreaded process running.

Figure 8-61 Replicat Information

Add Replicat ✕

1 — 2 — 3 — 4 — 5

Replicat Information Replicat Options Managed Options Parameter File Properties File

Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.

Replicat Type

☒ Classic Replicat ☐ Coordinated Replicat

Process Name
OCIOBSTR

Description:

Right Drawer Popup

Next ▶

3. Enter the **Replicat Options** and click **Next**.
 - **Replicat Trail:** Name of the required trail file. For sample trail, provide tr.
 - **Subdirectory:** Enter `GG_HOME/opt/AdapterExamples/trail/` if using the sample trail.
 - **Target:** OCI Object Storage
 - **Format:** Select the file format.

Figure 8-62 Replicat Options

Add Replicat [X]

Replicat Information Replicat Options Managed Options Parameter File **Properties File**

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name tr	Subdirectory /u01/ogg/opt/AdapterE	Encryption Profile LocalWallet

Begin
Position in Trail

Trail Position

Sequence Number 0	RBA Offset 0

Target

Target OCI Object Storage

Format

Format Parquet

◀ Back **Next ▶**

4. Leave **Managed Options** as is and click **Next**.

Figure 8-63 Managed Options

Add Replicat ×

Replicat Information ✓ Replicat Options ✓ **Managed Options** 3 Parameter File 4 Properties File 5

Use managed options to manage replicat start and auto-start options.

Profile Name: Default ▼ Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes Startup Delay Seconds

Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries ▼ ▲ Retry Delay Minutes Right Drawer Popup Retry Delay Seconds

Retries Window Hours Retries Window Minutes Retries Window Seconds

◀ Back **Next ▶**

5. Enter **Parameter File** details and click **Next**.

In the Parameter File, you can either specify source to target mapping or leave it as-is with a wildcard selection. If Coordinated Replicat is selected as the Replicat Type, an additional parameter needs to be provided:

```
TARGETDB LIBFILE libggjava.so SET property=<ggbd-deployment_home>/etc/
conf/ogg/your_replicat_name.properties
```

Figure 8-64 Parameter File

Add Replicat ×

Replicat Information
Replicat Options
Managed Options
4
Properties File

Use parameter file to configure replicat parameters and source to target mapping.

```
REPLICAT OCIOBSTR
MAP *.* , TARGET *.*;
```

Back
Next

6. In **Properties File**, update the properties marked as #TODO and click **Create and Run**.

Properties file for Replicat

Configuration to load GoldenGate trail operation records
 # into OCI Object storage by chaining
 # File writer handler -> OCI Event handler.
 # Note: Recommended to only edit the configuration marked as TODO

gg.target=oci

#TODO: format can be 'parquet' or 'orc' or one of the pluggable formatter types.
 Default is 'parquet'.
 gg.format=parquet

#TODO: Edit the OCI region

gg.eventhandler.oci.region=<oci-region>

#TODO: Edit the OCI compartment OCID

gg.eventhandler.oci.compartmentID=<oci-compartment-ocid>

#TODO: Edit the OCI bucket name

gg.eventhandler.oci.bucketMappingTemplate=<oci-bucket-name>

#TODO: Edit the OCI Config file path

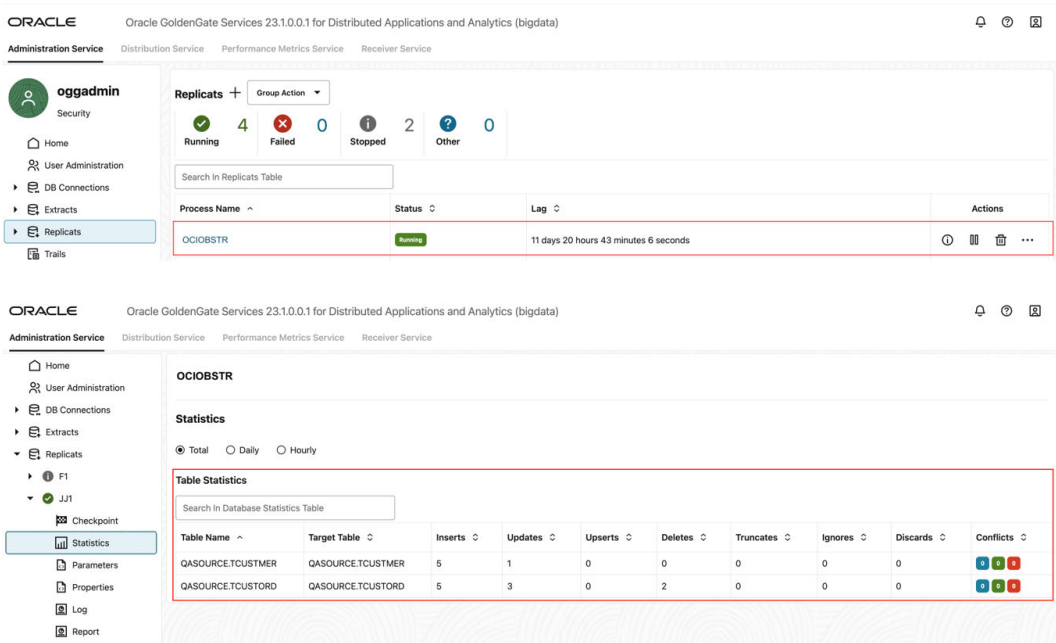
gg.eventhandler.oci.configFilePath=./oci/config

#TODO: Edit to include the OCI Java SDK.

gg.classpath= ./oci-java-sdk/lib/*:./oci-java-sdk/third-party/lib/*:/path/to/hadoop-deps:/path/to/parquet_deps/*

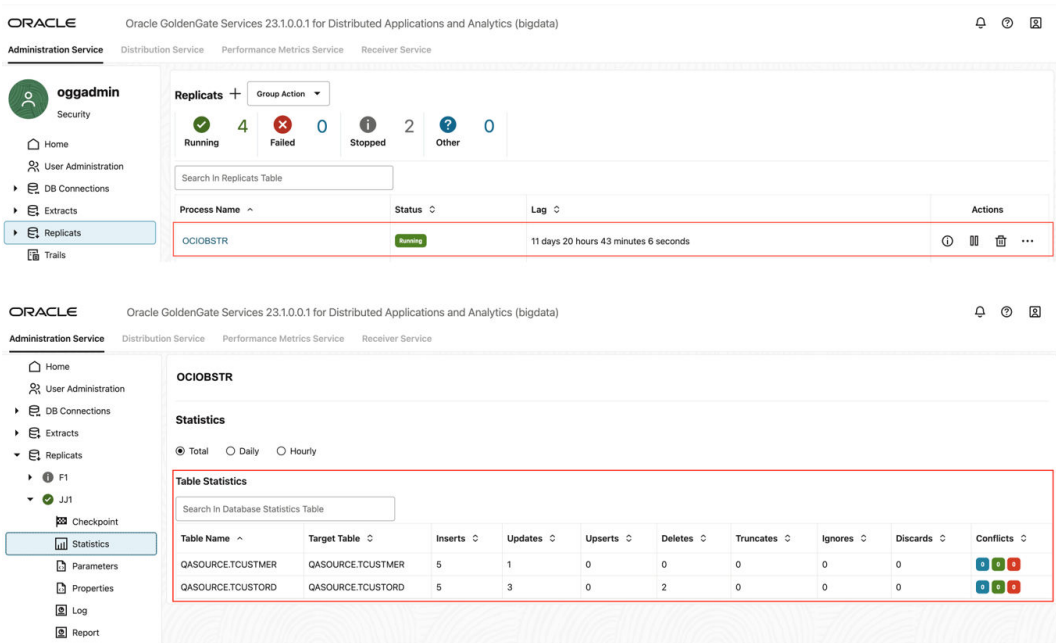
7. If replicat starts successfully, then it is in running state. Go to **action/details/statistics** to see the replication statistics.

Figure 8-65 Replication Statistics



8. Go to OCI console and check the bucket.

Figure 8-66 OCI Console



Note

- If target OCI Object Storage bucket does not exist, then it will be auto created by GG for DAA. You can use [Template Keywords](#) to dynamically assign OCI bucket names.
- OCI Object Storage Event Handler can be configured for proxy server. For more information, see [OCI Object Storage Event Handler](#).
- You can use different properties to control the behaviour of file writing. You can set file sizes, inactivity periods and more. You can get more details in the File Writer [blog post](#).

8.10 Realtime Parquet Ingestion into OCI Object Storage with Oracle GoldenGate for Distributed Applications and Analytics

Overview

This Quickstart covers a step-by-step process showing how to ingest parquet files into OCI Object Storage buckets in real-time with GoldenGate for Distributed Applications and Analytics (GG for DAA).

OCI Object Storage is a scalable, high-performance storage service provided by Oracle Cloud Infrastructure (OCI). It enables users to store and manage large amounts of unstructured data, such as images, videos, log files, backups, and other types of files.

GG for DAA OCI Object Storage handler works in conjunction with [File Writer Handler](#) and [Parquet Handler](#) (if parquet is required). File Writer Handler produces files locally, optionally Parquet Handler converts to parquet format and [OCI Object Storage Handler](#) loads into OCI Object Storage buckets.

- [Prerequisites](#)
- [Install Dependency Files](#)
- [Configure Credentials for Oracle Cloud Infrastructure](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.10.1 Prerequisites

To successfully complete this Quickstart, you must have the following:

- OCI Object Storage access

In this Quickstart, a sample trail file (named *tr*), which is shipped with GG for DAA is used. The sample trail file is located at `GG_HOME/opt/AdapterExamples/trail/` in your GG for DAA instance.

8.10.2 Install Dependency Files

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) uses client libraries in the replication process and these libraries need to be downloaded before setting up the replication process. You can use dependency downloader to download the client libraries. [Dependency Downloader](#) is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

GG for DAA uses a 3-step process to ingest parquet into OCI Object Storage buckets:

- Generating local files from trail files
- Converting local files to Parquet format
- Loading files into OCI Object Storage buckets

For generating local parquet files with GG for DAA, replicat uses [File Writer Handler](#) and [Parquet Handler](#). To load the parquet files into OCI Object Storage, GG for DAA uses [OCI Event Handler](#) in conjunction with File Writer and Parquet Event Handler.

GG for DAA uses 3 different set of client libraries to create parquet files and loading into OCI Object Storage:

1. In your GG for DAA VM, go to Dependency Downloader utility. It is located at `GG_HOME/opt/DependencyDownloader/`
2. Run `parquet.sh` and `hadoop.sh` with and `oracle_oci.sh` with the required versions.

Figure 8-67 Run `parquet.sh` and `hadoop.sh` with and `oracle_oci.sh` with the required versions

```

-bash-4.2$ pwd
/u01/app/ogg/opt/DependencyDownloader
-bash-4.2$ ls
aws.sh          docs          hadoop_cloudera.sh  internal_scripts  kafka_mapr.sh    project
bigquery.sh     elasticsearch_rest.sh  hadoop_hortonworks.sh  kafka.sh          mongodb.sh       snowflake.sh
cassandra.sh    elasticsearch_transport.sh  hadoop_mapr.sh        kafka_cloudera.sh  oracle_nosql_sdk.sh  synapse.sh
cassandra_dse.sh  gcs.sh         hbase.sh            kafka_confluent.sh  oracle_oci.sh       velocity.sh
config_proxy.sh  hadoop.sh      hbase_cloudera.sh   kafka_confluent_protobuf.sh  orc.sh
dependencies     hadoop_azure_cloudera.sh  hbase_hortonworks.sh  kafka_hortonworks.sh  parquet.sh
-bash-4.2$ ./parquet.sh 1.12.2

labkeys -- ssh -i id_rsa opc@141.144.231.135 -- 150x33
-bash-4.2$ ls
aws.sh          docs          hadoop_cloudera.sh  internal_scripts  kafka_mapr.sh    project
bigquery.sh     elasticsearch_rest.sh  hadoop_hortonworks.sh  kafka.sh          mongodb.sh       snowflake.sh
cassandra.sh    elasticsearch_transport.sh  hadoop_mapr.sh        kafka_cloudera.sh  oracle_nosql_sdk.sh  synapse.sh
cassandra_dse.sh  gcs.sh         hbase.sh            kafka_confluent.sh  oracle_oci.sh       velocity.sh
config_proxy.sh  hadoop.sh      hbase_cloudera.sh   kafka_confluent_protobuf.sh  orc.sh
dependencies     hadoop_azure_cloudera.sh  hbase_hortonworks.sh  kafka_hortonworks.sh  parquet.sh
-bash-4.2$ ./hadoop.sh 3.1.1

-bash-4.2$ cd /u01/app/ogg/opt/DependencyDownloader/
-bash-4.2$ ls
aws          cassandra_capture_iv.sh  docs          hadoop_azure_cloudera.sh  hbase_hortonworks.sh  internal_scripts  kafka_hortonworks.sh  oracle_oci.sh  velocity.sh
aws_vml     cassandra_restore_iv.sh  download_dependencies.sh  hadoop_cloudera.sh      hadoop_hortonworks.sh  kafka.sh          mongodb.sh           orc.sh
aws_vml     cassandra_dse.sh        elasticsearch_rest.sh     hadoop_mapr.sh         hadoop_mapr.sh        kafka_cloudera.sh  mongo_4.6.0_dependencies.zip  project
aws_vml     config_proxy.sh         elasticsearch_transport.sh  hadoop_mapr.sh         hadoop_mapr.sh        kafka_confluent.sh  oracle_nosql_sdk.sh  snowflake.sh
bigquery.sh  depend.sh              gcs.sh          hbase.sh               hbase_cloudera.sh    kafka_confluent_protobuf.sh  oracle_oci.sh  synapse.sh
cassandra.sh  dependencies            hadoop.sh       hbase_cloudera.sh      hbase_hortonworks.sh  kafka_hortonworks.sh  parquet.sh
-bash-4.2$ ./oracle_oci.sh 3.2.0

-bash-4.2$ cd dependencies/
-bash-4.2$ ls
aws.sh 1.12.369 hadoop 3.3.0 oracle_oci 3.2.0 parquet 1.12.3
-bash-4.2$

```

3. 3 directories are created in `GG_HOME/opt/DependencyDownloader/dependencies`. Note the directories. For example:
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/oracle_oci_3.2.0/*`
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/hadoop_3.4.0/*`
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/parquet_1.12.3/*`

8.10.3 Configure Credentials for Oracle Cloud Infrastructure

You need to create a configuration file to authenticate into OCI. The ideal configuration file include user, fingerprint and key_file, tenancy, and region with their respective values. The default configuration file name and location is ~/.oci/config. For more information, see [Required Keys and OCIDs](#) documentation.

Sample Config File

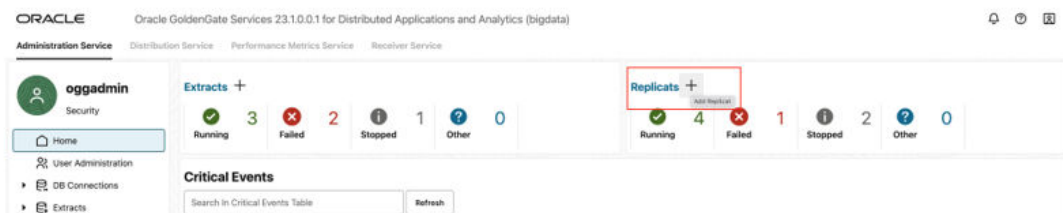
```
[DEFAULT]
user=<your_user_ocid>
fingerprint=<your_fingerprint>
key_file=~/.oci/oci_api_key.pem #path-to_your_key_file
tenancy=<your_tenancy_ocid>
```

8.10.4 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

To create a replicat in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA):

1. In the GG for DAA UI, in the **Administration Service** tab, click the + sign to add a replicat.

Figure 8-68 Click the Administration Service tab



2. Select the **Classic Replicat** Replicat Type and click **Next**. There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-69 Add Replicat

Add Replicat

1 Replicat Information 2 Replicat Options 3 Managed Options 4 Parameter File 5 Properties File

Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.

Replicat Type

☒ Classic Replicat ☐ Coordinated Replicat

Process Name: OCIDBSTR

Description:

Run Dumper PopUp

Next

3. Enter the Replicat information, and click **Next**:
 - a. **Replicat Trail**: Name of the required trail file. For sample trail, provide tr.
 - b. **Subdirectory**: Enter GG_HOME/opt/AdapterExamples/trail/ if using the sample trail.
 - c. **Target**: OCI Object Storage

Figure 8-70 Replicat Options

Add Replicat

1 Replicat Information 2 **Replicat Options** 3 Managed Options 4 Parameter File 5 Properties File

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: tr Subdirectory: Encryption Profile: LocalWallet

Begin Position in Trail: Position in Trail

Trail Position

Sequence Number: 0 RSA Offset: 0

Target:

Target: OCI Object Storage
Big Data target entity

Back Next

4. Leave **Managed Options** as is and click **Next**.

Figure 8-71 Managed Options

5. Enter **Parameter File** details and click **Next**. In the Parameter File, you can either specify source to target mapping or leave it as-is with a wildcard selection. If Coordinated Replicat is selected as the Replicat Type, then an additional parameter needs to be provided:
TARGETDB LIBFILE libggjava.so SET property=<ggbd-deployment_home>/etc/
conf/ogg/your_replicat_name.properties

Figure 8-72 Parameter File

Add Replicat

Replicat Information Replicat Options Managed Options **Parameter File** Properties File

Use parameter file to configure replicat parameters and source to target mapping.

REPLICAT OCIOBSTR
MAP *.*, TARGET *.*;

Right Drawer Popup

Back Next

6. In the Properties File, remove all the pre-configured properties; but not the first row marked with the replicat name (# Properties file for Replicat <replicat_name>). Copy and paste below property list into properties file, update the properties marked as #TODO and click **Create and Run**.

#The File Writer Handler - no need to change

```
gg.handlerlist=filewriter
gg.handler.filewriter.type=filewriter
gg.handler.filewriter.mode=op
gg.handler.filewriter.pathMappingTemplate=./dirout
gg.handler.filewriter.stateFileDirectory=./dirsta
gg.handler.filewriter.fileRollInterval=7m
gg.handler.filewriter.inactivityRollInterval=5s
gg.handler.filewriter.fileWriteActiveSuffix=.tmp
gg.handler.filewriter.finalizeAction=delete
```

Avro OCF - no need to change

```
gg.handler.filewriter.format=avro_row_ocf
gg.handler.filewriter.fileNameMappingTemplate=${groupName}_${fullyQualifiedTableName}_${currentTimestamp}.avro
gg.handler.filewriter.format.pkUpdateHandling=delete-insert
gg.handler.filewriter.format.metaColumnsTemplate=${optype},${position}
gg.handler.filewriter.format.iso8601Format=false
gg.handler.filewriter.partitionByTable=true
gg.handler.filewriter.rollOnShutdown=true
```

#The Parquet Event Handler - no need to change

```
gg.handler.filewriter.eventHandler=parquet
gg.eventhandler.parquet.type=parquet
gg.eventhandler.parquet.pathMappingTemplate=./dirparquet
```

```
gg.eventhandler.parquet.fileNameMappingTemplate=${groupName}_${
fullyQualifiedTableName}_${currentTimestamp}.parquet
gg.eventhandler.parquet.writeToHDFS=false
gg.eventhandler.parquet.finalizeAction=delete
```

#TODO Select OCI Event Handler - no need to change

```
gg.eventhandler.parquet.eventHandler=oci
```

#TODO Set OCI Event Handler - please update as needed

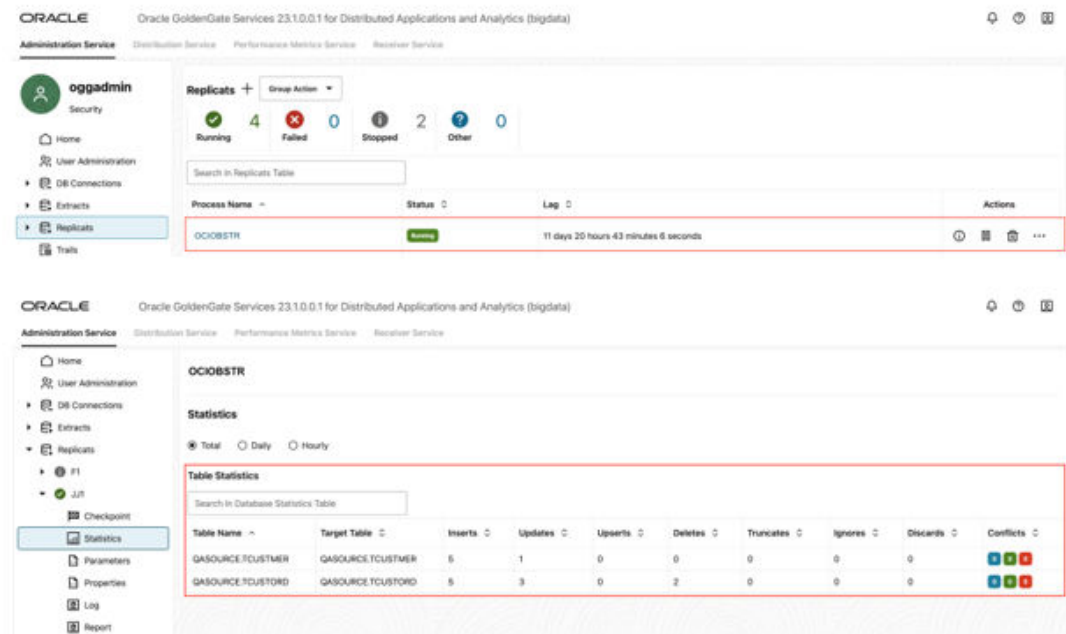
```
gg.eventhandler.oci.type=oci
gg.eventhandler.oci.region=<your_bucket_region>
gg.eventhandler.oci.compartmentID=<your_compartment_ocid>
gg.eventhandler.oci.bucketMappingTemplate=<your_bucket_name>
gg.eventhandler.oci.pathMappingTemplate=${schemaName}
gg.eventhandler.oci.fileNameMappingTemplate=${tableName}_${currentTimestamp}.parquet
gg.eventhandler.oci.finalizeAction=NONE
gg.eventhandler.oci.configFilePath=path_to_oci_config_file_from_step2
#TODO: Edit to include the OCI Java SDK.
```

#TODO Set the classpath to the paths you noted in step1

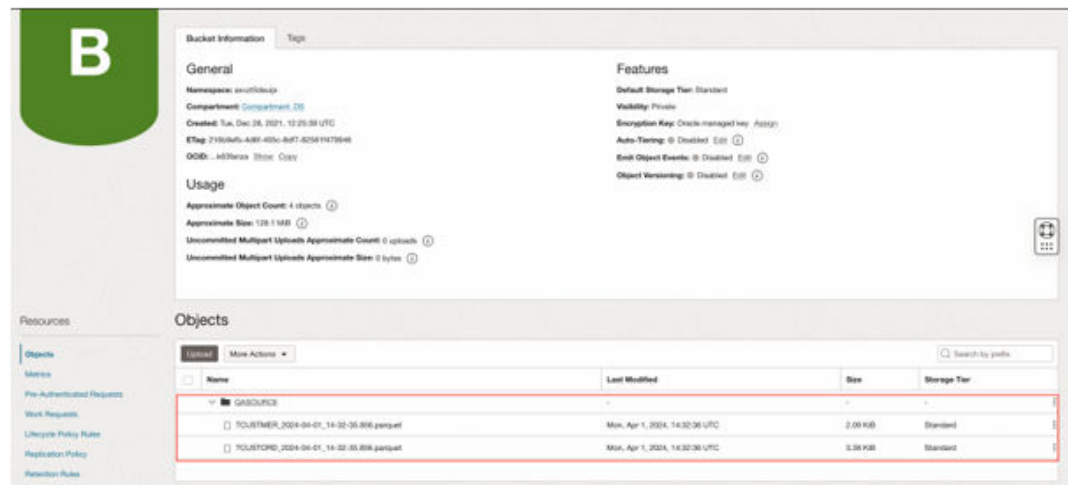
```
gg.classpath=path_to/ gcs_12.29.1/: path_to /hadoop_3.4.0/:path_to/parquet_1.12.3/*
jvm.bootoptions=-Xmx512m -Xms32m
```

7. If replicat starts successfully, then it is in running state. You can go to Replicats/Statistics to see the replication statistics.

Figure 8-73 Replication Statistics



8. Go to the OCI console and check the bucket.

Figure 8-74 OCI Console**Note**

- If target OCI Object Storage bucket does not exist, it will be auto created by GG for DAA. You can use [Template Keywords](#) to dynamically assign the container names.
- OCI Object Storage Event Handler can be configured for proxy server. For more information, see [OCI Object Storage Event Handler](#).
- You can use different properties to control the behaviour of file writing. You can set file sizes, inactivity periods and more. You can get more details in the [File Writer blog post](#).

8.11 Realtime Message Ingestion to OCI Streaming with Oracle GoldenGate for Distributed Applications and Analytics

Overview

This Quickstart covers a step-by-step process showing how to ingest messages to OCI Streaming in real-time with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

Oracle Cloud Infrastructure (OCI) Streaming provides a fully managed, scalable, and durable messaging solution for ingesting continuous, high-volume streams of data that you can consume and process in real-time. OCI Streaming supports Kafka APIs.

GG for DAA connects to OCI Streaming with [Kafka Handler](#). GG for DAA reads the source operations from the trail file, formats them, maps to OCI Streaming streams and delivers.

- [Prerequisites](#)
- [Install Dependency Files](#)
- [Create Kafka Producer Properties](#)
- [Create a Replicat in GG for DAA](#)

8.11.1 Prerequisites

To successfully complete this Quickstart, you must have the following:

- OCI Streaming Stream Pools
- OCI Auth Token

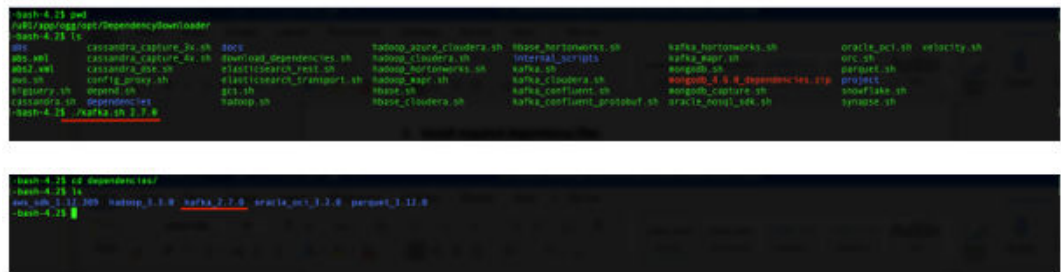
In this Quickstart, a sample trail file (named *tr*) which is shipped with GG for DAA is used. The sample trail file is located at `GG_HOME/opt/AdapterExamples/trail/` in your GG for DAA instance.

8.11.2 Install Dependency Files

GG for DAA uses Java SDK provided by OCI Streaming. You can download the SDKs using [Dependency Downloader](#) utility shipped with GG for DAA. Dependency downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

1. In your GG for DAA VM, go to Dependency Downloader utility. It is located at `GG_HOME/opt/DependencyDownloader/` and locate `kafka.sh`.
2. Run `kafka.sh` with the required version. You can check the version and reported vulnerabilities in [Maven Central](#). This document uses 2.7.0.

Figure 8-75 Run kafka.sh



3. A directory is created in `GG_HOME/opt/DependencyDownloader/dependencies`. Note the directories. For example: `/u01/app/ogg/opt/DependencyDownloader/dependencies/kafka_2.7.0`

8.11.3 Create Kafka Producer Properties

GG for DAA must access a Kafka producer configuration file to publish messages to OCI Streaming. The Kafka producer configuration file contains kafka connection settings provided by OCI Streaming. To get OCI Streaming Kafka connection settings, go to **Analytics&AI/Streaming/Stream Pools/Stream Pool Details/Kafka Connection** Settings. You also need to create an [AUTH TOKEN](#).

In your GG for DAA instance, create a Kafka producer config file for OCI Streaming.

Sample Config File

```
bootstrap.servers=cell-1.streaming.us-phoenix-1.oci.oraclecloud.com:9092
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
value.serializer=org.apache.kafka.common.serialization.ByteArraySerializer
key.serializer=org.apache.kafka.common.serialization.ByteArraySerializer
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="paasdevgo/oracleidentitycloudservice/user.name@oracle.com/"
```

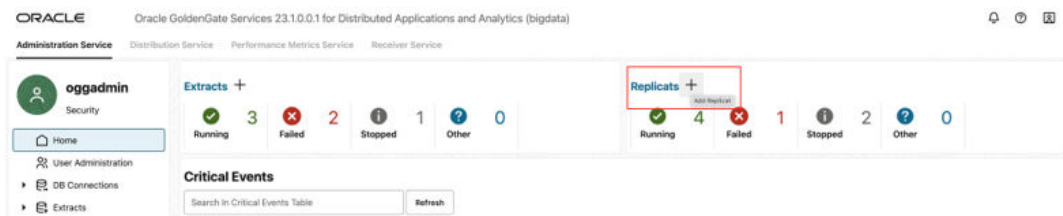
```
ocid1.streampool.oc1.phx.amaaaaaa3p5c3vqa4hfyl7uv465pay4audmoajughhxlsqj7afc2an5u3xaq"
password="YOUR-AUTH-TOKEN";
```

8.11.4 Create a Replicat in GG for DAA

To create a replicat in GG for DAA:

1. In the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) UI, in the **Administration Service** tab, click the **+** sign to add a replicat.

Figure 8-76 Click the Administration Service tab



2. Select the **Classic Replicat** Replicat Type and click **Next**. There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-77 Add Replicat

Add Replicat

1 Replicat Information 2 Replicat Options 3 Managed Options 4 Parameter File 5 Properties File

Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.

Replicat Type

☒ Classic Replicat ☐ Coordinated Replicat

Process Name
OCISTR

Description:

Next

3. Enter the Replicat information, and click **Next**:

- a. **Replicat Trail:** Name of the required trail file. For sample trail, provide tr.
- b. **Subdirectory:** Enter GG_HOME/opt/AdapterExamples/trail/ if using the sample trail.
- c. **Target:** Kafka

Figure 8-78 Replicat Options

Add Replicat [X]

1 Replicat Information 2 Replicat Options 3 Managed Options 4 Parameter File 5 Properties File

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: tr Subdirectory: u01/app/ogg/opt/Adapt Encryption Profile: LocalWallet

Begin Position in Trail: [Begin]

Trail Position

Sequence Number: 0 RBA Offset: 0

Target:

Target: Kafka

Kafka Connect:

[Back] [Next]

4. Leave **Managed Options** as is and click **Next**.

Figure 8-79 Managed Options

Add Replicat

Replicat Information Replicat Options **Managed Options** Parameter File Properties File

Use managed options to manage replicat start and auto-start options.

Profile Name: Default Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes 0 Startup Delay Seconds 0

Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries 9 Retry Delay Minutes 0 Retry Delay Seconds 0

Retries Window Hours 0 Retries Window Minutes 0 Retries Window Seconds 0

Back Next

5. Enter **Parameter File** details and click **Next**. In the Parameter File, you can either specify source to target mapping or leave it as-is with a wildcard selection.

Figure 8-80 Parameter File

Add Replicat

Replicat Information Replicat Options Managed Options **Parameter File** Properties File

Use parameter file to configure replicat parameters and source to target mapping.

REPLICAT OCISTR
MAP *.* TARGET *.*;

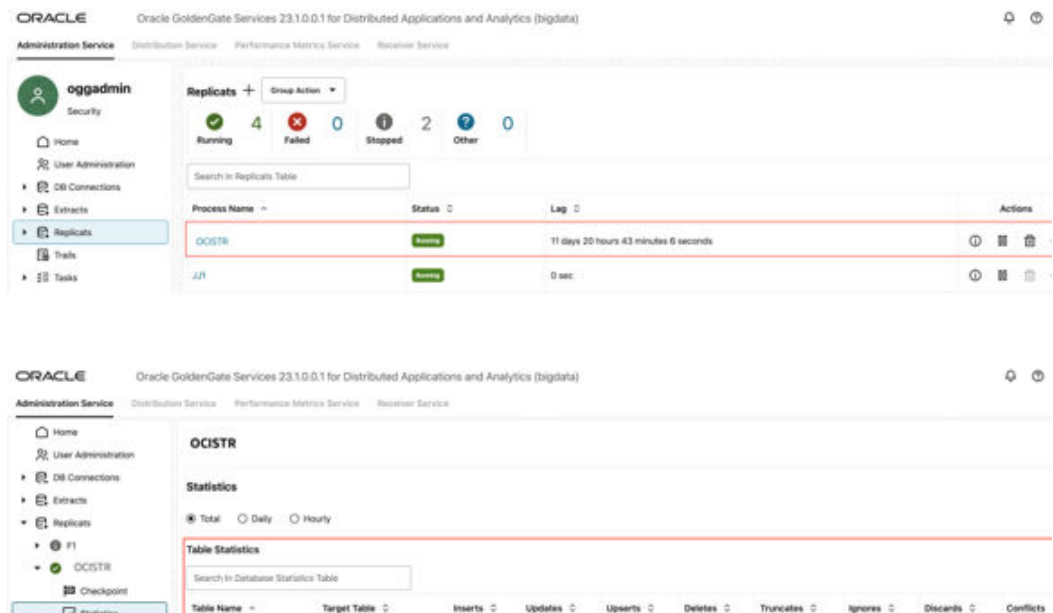
Back Next

6. In the Properties File, update the properties marked as TODO and click **Create and Run**.

```
#Kafka Handler Template
gg.handlerlist=kafkahandler
gg.handler.kafkahandler.type=kafka
#TODO: Set the name of the Kafka producer properties file.
gg.handler.kafkahandler.kafkaProducerConfigFile=/path_to/producer.properties
#TODO: Set the template for resolving the topic name.
gg.handler.kafkahandler.topicMappingTemplate=<target_stream_name>
gg.handler.kafkahandler.keyMappingTemplate=${primaryKeys}
gg.handler.kafkahandler.mode=op
gg.handler.kafkahandler.format=json
gg.handler.kafkahandler.format.metaColumnsTemplate=${objectname[table]},$
{optype[op_type]},${timestamp[op_ts]},${currenttimestamp[current_ts]},$
{position[pos]}
#TODO: Set the location of the Kafka client libraries.
gg.classpath=path_to/dependencies/kafka_2.7.0/*
jvm.bootoptions=-Xmx512m -Xms32m
```

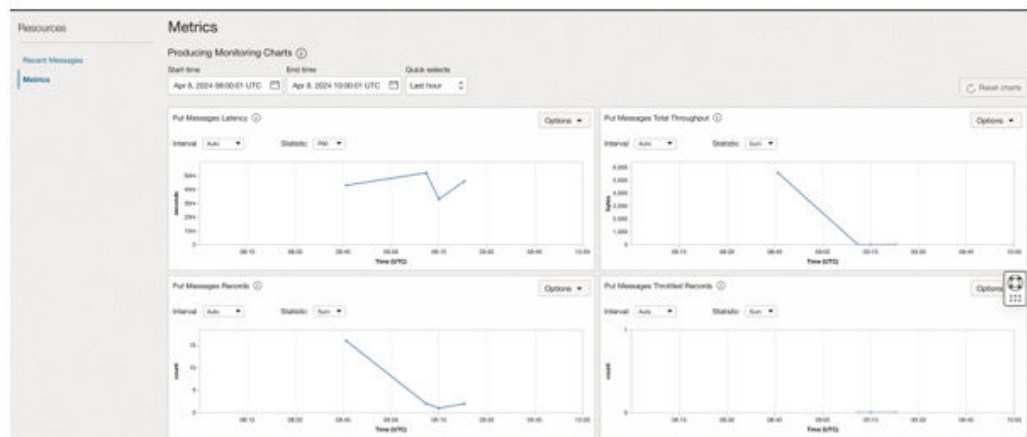
7. If replicat starts successfully, then it is in running state. You can go to Replicats/Statistics to see the replication statistics.

Figure 8-81 Replicat Statistics



8. Go to the OCI console and check the statistics.

Figure 8-82 OCI Streaming console



For more details about OCI Streaming replication, see [OCI Streaming](#).

Note

- If target kafka topic does not exist, then it is auto created by GG for DAA if **Auto topic create** is selected in OCI streaming Kafka connection settings. See [Template Keywords](#) to dynamically assign the topic names.
- You can refer to [this blog](#) for improving the performance of the OCI Streaming replication.

8.12 Realtime Message Ingestion to Azure Event Hubs with Oracle GoldenGate for Distributed Applications and Analytics

Overview

This Quickstart covers a step-by-step processes showing how to ingest messages to Azure Event Hub in real-time with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

Azure Event Hubs is a cloud native data streaming service that can stream high volumes of messages. Azure Event Hubs provides an Apache Kafka endpoint on an event hub, which enables users to connect to the event hub using the Kafka protocol.

GG for DAA connects to Azure Event Hub Apache Kafka Endpoint with [Kafka Handler](#). GG for DAA reads the source operations from the trail file, formats them, maps to Azure Event Hubs and delivers.

- [Prerequisites](#)
- [Install Dependency Files](#)
- [Create a producer.properties for Azure Event Hubs](#)
- [Create a Replicat in Oracle GoldenGate for Distirbuted Applications and Analytics](#)

8.12.1 Prerequisites

To successfully complete this Quickstart, you must have the following:

- An Azure Event Hubs Namespace
- Shared Access Policies for your Azure Event Hubs Namespace

In this Quickstart, a sample trail file (named `tr`), which is shipped with GG for DAA is used. The sample trail file is located at `GG_HOME/opt/AdapterExamples/trail/` in your GG for DAA instance.

8.12.2 Install Dependency Files

GG for DAA uses Apache Kafka client libraries for Azure Event Hubs. You can download the SDKs using [Dependency Downloader](#) utility shipped with GG for DAA. Dependency downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

1. In your GG for DAA VM, go to dependency downloader utility. It is located at `GG_HOME/opt/DependencyDownloader/` and locate `kafka.sh`.
2. Run `kafka.sh` with the required version. You can check the version and reported vulnerabilities in [Maven Central](#). This document uses 3.7.0 which is the latest version when this quick start is published.

Figure 8-83 Run kafka.sh

```

-bash-4.2$ ls
ls      am02.xml  dq      cassandra_capture_dir.sh  dependencies  hadoop.sh      Hbase.sh      kafka_client.sh      mongosh.sh
ls.xml  am02     dq.xml  cassandra_cdh.sh         download_dependencies.sh  hadoop_azure_client.sh  kafka_client.sh      mongo_capture.sh
ls      am02.xml  cassandra.sh  config_proxy.sh          elasticsearch_java.sh    hadoop_client.sh  kafka_client_protobuf.sh  oracle_node.sh
ls.xml  am02     cassandra_capture_3x.sh  delta                  gcs.sh              hadoop_hortonworks.sh  internal_scripts      kafka_hortonworks.sh
ls      hbase.sh  cassandra_capture_4x.sh  delta.xml              gcs.sh              hadoop_mapr.sh        kafka.sh            kafka_mapr.sh
-bash-4.2$ ./kafka.sh 3.7.0

```

3. A new directory is created in /u01/app/ogg/opt/DependencyDownloader/dependencies/kafka_3.7.0.

8.12.3 Create a producer.properties for Azure Event Hubs

In GG for DAA instance, create a file called `producer.properties` and copy the following configuration.

```
bootstrap.servers=<namespace>.servicebus.windows.net:9093
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password=" Endpoint=sb://
mynamespace.servicebus.windows.net/;SharedAccessKey=RootManageSharedAccessKey;SharedA
ccessKey=XXXXXXXXXXXXXXXXX";
value.serializer = org.apache.kafka.common.serialization.ByteArraySerializer
key.serializer = org.apache.kafka.common.serialization.ByteArraySerializer
```

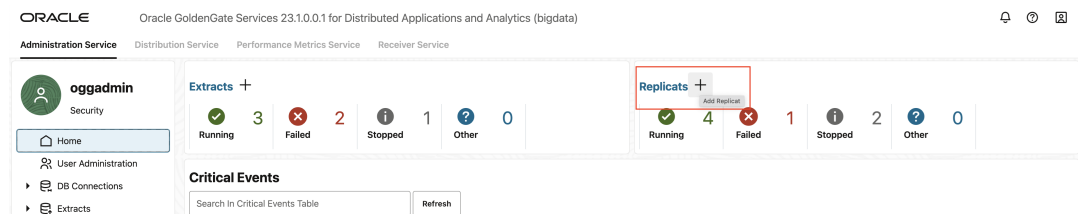
For more information, see [Azure Event Hub Shared Access Key](#).

8.12.4 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

To create a replicat in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA):

1. In the GG for DAA UI, in the **Administration Service** tab, click the **+** sign to add a replicat.

Figure 8-84 Click + in the Administration Service tab.



2. Select the **Classic Replicat** Replicat Type and click **Next** There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-85 Add Replicat

3. Enter the basic information, and click **Next**:
 - a. **Replicat Trail** : Name of the required trail file (if using sample trail, provide as *tr*)
 - b. **Subdirectory**: Enter GG_HOME/opt/AdapterExamples/trail/ if using the sample trail.
 - c. **Target**: Kafka

Figure 8-86 Replicat Options

Add Replicat ×

1 ✓ 2 ● 3 ○ 4 ○ 5 ○

Replicat Information Replicat Options Managed Options Parameter File Properties File

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name tr	Subdirectory /u01/app/ogg/opt/Adap	Encryption Profile LocalWallet

Begin
Position in Trail

Trail Position

Sequence Number 0	RSA Offset 0

Target:

Target: Kafka

Kafka Connect:
☐

◀ Back Next ▶

4. Leave **Managed Options** as is and click **Next**.

Figure 8-87 Managed Options

Add Replicat [X]

Replicat Information Replicat Options **Managed Options** Parameter File Properties File

Use managed options to manage replicat start and auto-start options.

Profile Name: Default Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes 0 Startup Delay Seconds 0

Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries 9 Retries Window Hours 0 Retries Window Minutes 0 Retries Window Seconds 0

Right Drawer Popup

Back Next

5. Enter **Parameter File** details and click **Next**. In the Parameter File, you can specify source to target mapping or leave it as-is with a wildcard selection.

Figure 8-88 Parameter File

Add Replicat

Replicat Information Replicat Options Managed Options **Parameter File** Properties File

Use parameter file to configure replicat parameters and source to target mapping.

```
REPLICAT EVNTHUB
MAP *,* TARGET *,*;
```

Back Next

6. In the Properties file, update the properties marked as **TODO** and click **Create and Run**.

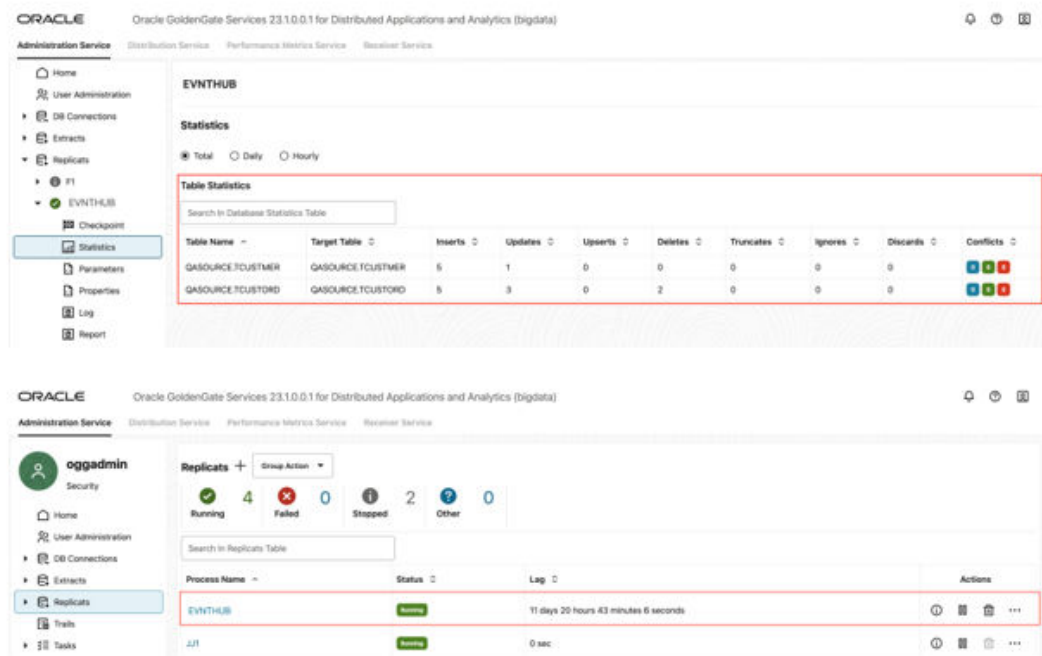
```
#Kafka Handler Template
gg.handlerlist=kafkahandler
gg.handler.kafkahandler.type=kafka
#TODO: Set the name of the Kafka producer properties file.
gg.handler.kafkahandler.kafkaProducerConfigFile=/path_to/producer.properties
#TODO: Set the template for resolving the topic name.
gg.handler.kafkahandler.topicMappingTemplate=<target_event_hub_name>
gg.handler.kafkahandler.keyMappingTemplate=${primaryKeys}
gg.handler.kafkahandler.mode=op
gg.handler.kafkahandler.format=json
gg.handler.kafkahandler.format.metaColumnsTemplate=${objectname[table]}, $
{optype[op_type]}, ${timestamp[op_ts]}, ${currenttimestamp[current_ts]}, $
{position[pos]}
#TODO: Set the location of the Kafka client libraries.
gg.classpath=path_to/dependencies/kafka_3.7.0/*
jvm.bootoptions=-Xmx512m -Xms32m
```

GG for DAA supports dynamic topic mapping by template keywords. For example, if you assign topicMappingTemplate as \${tablename}, GG for DAA will create an Event Hub with the source table name, per each source table and will map the events to these topics.

Oracle recommends to use keyMappingTemplate=\${primaryKeys}, GG for DAA sends the source operations with the same pk to the same partition. This will guarantee maintaining the order of the source operations while delivering to Azure Event Hubs.

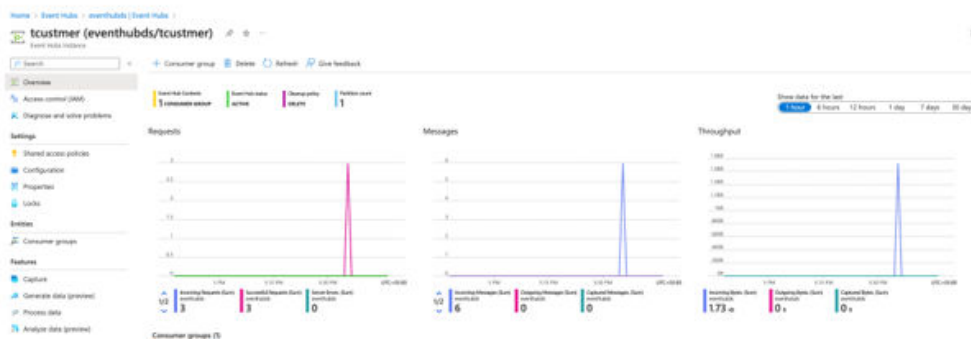
7. If replicat starts successfully, it will be in running state. You can go to action/details/statistics to see the replication statistics.

Figure 8-89 Replicat Statistics



- You can go to Azure Event Hub console and check the statistics.

Figure 8-90 Azure Event Hub console



For more details about Azure Event Hub replication, see [Apache Kafka](#).

Note

- If target kafka topic does not exist, then it is auto created by GG for DAA if the auto topic create is selected in OCI Streaming Kafka connection settings. You can use [Template Keywords](#) to dynamically assign topic names.
- See [blog](#) for improving the performance of the OCI Streaming replication.

8.13 Realtime Data Ingestion into GCP BigQuery with Oracle GoldenGate for Distributed Applications and Analytics

Overview

This Quickstart covers a step-by-step process showing how to ingest real-time data into GCP BigQuery with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

GCP BigQuery is a cloud-based data warehousing platform that provides a fully managed service for storing, processing, and analysing data.

GG for DAA supports [stage and merge](#) and [streaming api](#) designs. In stage and merge, the change data is staged in a temporary location in microbatches and eventually merged into to the target table. GCS is used as the staging location.

All replication process is automatically handled by [GoldenGate for Distributed Applications and Analytics \(GG for DAA\) GCP BigQuery Stage and Merge Handler](#).

- [Prerequisites for Google Cloud Platform BigQuery Stage and Merge](#)
- [Install Dependency Files](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.13.1 Prerequisites for Google Cloud Platform BigQuery Stage and Merge

- Google Cloud Platform (GCP) account set up.
- A Google Cloud Platform (GCP) [service account key](#) with relevant [BigQuery Permissions](#). Copy your GCP service account key to a directory on your Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) Server.
- A Google Cloud Storage bucket with [relevant permissions](#). Ensure that the GCS bucket and the BigQuery dataset exist in the same location or region.
- Target BigQuery tables can be created before configuring the replicat. If necessary permissions are provided, then GG for DAA can auto create the target BigQuery tables.

In this quick start, we will use a sample trail file (named tr) which is shipped with GG for DAA. If you want to continue with sample trail file, it is located at GG_HOME/opt/AdapterExamples/trail/ in your GG for DAA instance.

GG for DAA will create the tables automatically.

8.13.2 Install Dependency Files

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) uses Java SDK provided by Google. You can download the SDKs using [Dependency Downloader](#) utility shipped with GG for DAA. Dependency downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

1. In your GG for DAA VM, go to dependency downloader utility. It is located at GG_HOME/opt/DependencyDownloader/.
2. Run [gcs.sh](#) and [bigquery.sh](#) with the required versions.

Figure 8-91 Run gcs.sh and bigquery.sh with the required versions

```

bash-4.2$ pwd
/u01/app/ogg/opt/DependencyDownloader
bash-4.2$ ls
bigquery.sh  cassandra_capture_dse.sh  elasticsearch_rest.sh  hadoop_cloudera.sh  hbase_hortonworks.sh  kafka_confluent_protobuf.sh  oracle_noq1_sds.sh  redis.sh
cassandra.sh  cassandra_dse.sh  elasticsearch_transport.sh  hadoop_hortonworks.sh  kafka_hortonworks.sh  kafka_confluent_protobuf.sh  oracle_sds.sh  snowflake.sh
cassandra_capture.sh  config_bigquery.sh  gcs.sh  hadoop_hdp.sh  kafka_hdp.sh  kafka_hdp.sh  arc.sh  s3capture.sh
cassandra_capture_dse.sh  docx  hadoop_azure_cloudera.sh  hbase_cloudera.sh  kafka_cloudera.sh  kafka_confluent.sh  mssql_capture.sh  velocity.sh
bash-4.2$

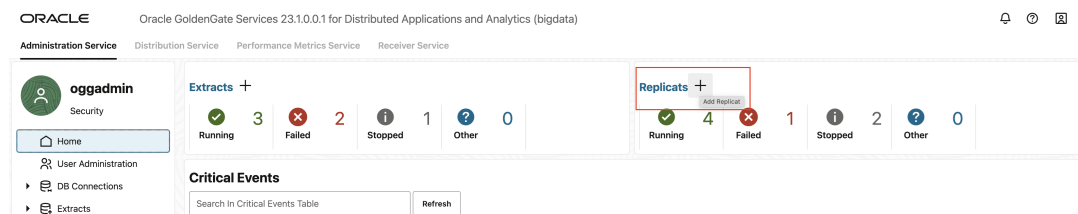
```

3. 2 directories are created in GG_HOME/opt/DependencyDownloader/dependencies. Make a note of the directories:
 - /u01/app/ogg/opt/DependencyDownloader/dependencies/bigquery_1.111.1
 - /u01/app/ogg/opt/DependencyDownloader/dependencies/gcs_1.113.9

8.13.3 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

To create a replicat in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA):

1. In the **Administration Service** tab, click the + sign to add a replicat.

Figure 8-92 Click + in the Administration Service tab.

2. Select the **Classic Replicat** Replicat Type and click **Next**. There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-93 Add Replicat

Add Replicat

1 Replicat Information 2 Replicat Options 3 Managed Options 4 Parameter File 5 Properties File

Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.

Replicat Type

☒ Classic Replicat ☐ Coordinated Replicat

Process Name
GCPBQ

Description:

Next

3. Enter the Replicat options, and click **Next**:
 - a. **Trail Name**: Name of the required trail file (if using sample trail, provide as tr)
 - b. **Subdirectory**: Provide as GG_HOME/opt/AdapterExamples/trail/ if using the sample trail.
 - c. **Target**: Google BigQuery
- Enable Stage and Merge and select Google Cloud Storage as Available staging location.

Figure 8-94 Replicat Options

Add Replicat [X]

Replicat Information Replicat Options Managed Options Parameter File Properties File

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: Subdirectory: Encryption Profile:

Begin Position in Trail:

Trail Position

Sequence Number: RBA Offset:

Target:

Target:

Stage and Merge using external object storage: ☒ Available staging locations:

Back Next

4. Leave **Managed Options** as is and click **Next**.

Figure 8-95 Managed Options

Add Replicat

Replicat Information Replicat Options **Managed Options** Parameter File Properties File

Use managed options to manage replicat start and auto-start options.

Profile Name: **Default** Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes: 0 Startup Delay Seconds: 0

Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries: 9 Retries Delay Minutes: 0 Retries Delay Seconds: 0

Retries Window Hours: 0 Retries Window Minutes: 0 Retries Window Seconds: 0

Back **Next**

5. Enter **Parameter File** details and click **Next**. In the Parameter File, you can specify source to target mapping. If you are using the sample trail file (tr), then enter as follows: MAP QASOURCE.*, TARGET <your_schema_name>.*;
If Coordinated Replicat is selected as the Replicat Type, an additional parameter needs to be provided: TARGETDB LIBFILE libggjava.so SET property=<ggbd-deployment_home>/etc/conf/ogg/your_replicat_name.properties

Figure 8-96 Parameter File

Add Replicat

Replicat Information Replicat Options Managed Options **Parameter File** Properties File

Use parameter file to configure replicat parameters and source to target mapping.

```
REPLICAT GCPBQ
MAP +.*, TARGET +.*;
```

Back Next

6. In the next screen, you need to update the properties only tagged as **TODO** and click **Create and Run**.

Provide your GCS bucket name:

#TODO: Edit the GCS bucket name
 gg.eventhandler.gcs.bucketMappingTemplate=<gcs-bucket-name>

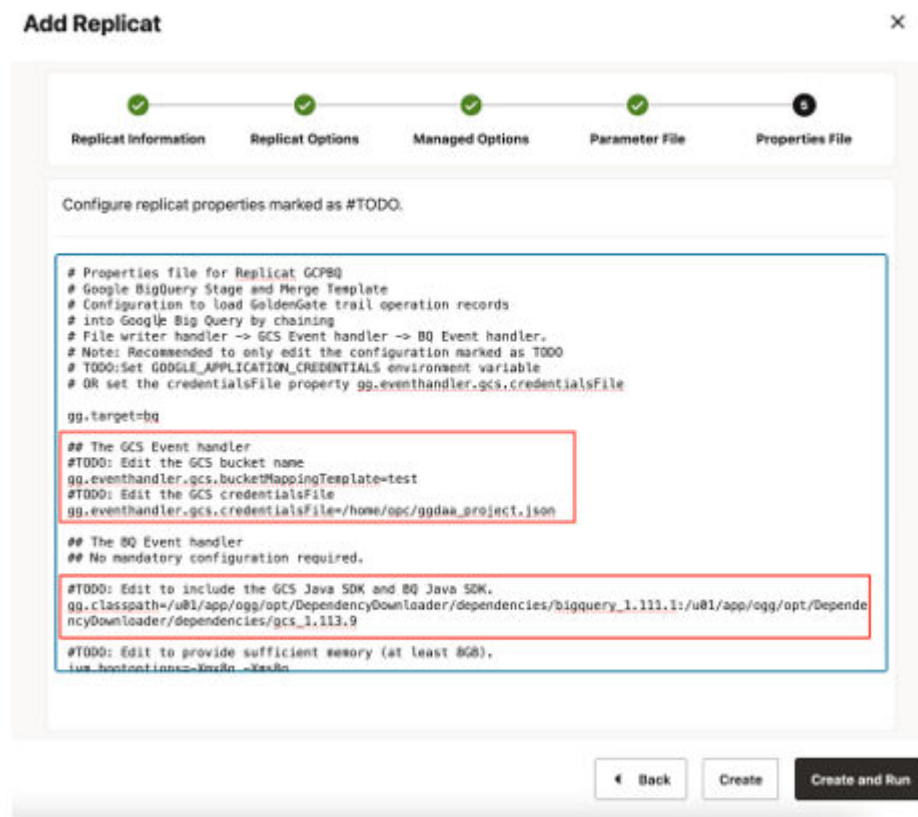
Provide path to your GCP service account key:

#TODO: Edit the GCS credentialsFile
 gg.eventhandler.gcs.credentialsFile=/path/to/gcp/credentialsFile

Provide path to dependency jar files that you downloaded in prerequisites:

#TODO: Edit to include the GCS Java SDK and BQ Java SDK.
 gg.classpath=/path/to/gcs_dependencies/*:/path/to/bq_dependencies/*

Figure 8-97 Properties File



For more information about replication configuration, see [Google BigQuery Stage and Merge](#).

7. If replicat starts successfully, it will be in running state. You can go to action/details/statistics to see the replication statistics.

Figure 8-98 Replication Statistics

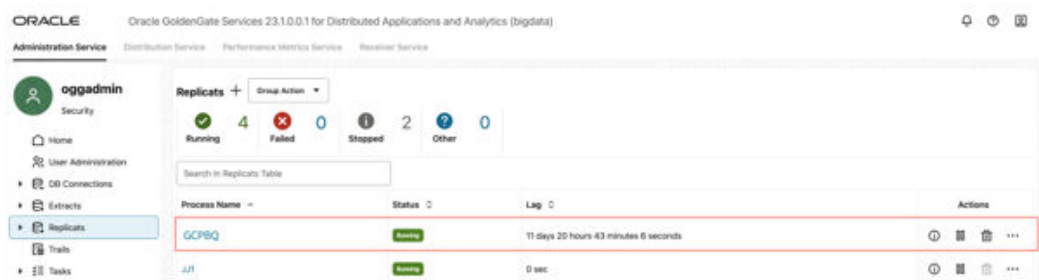


Figure 8-99 GCPBQ Statistics

Table Name	Target Table	Inserts	Updates	Upperts	Deletes	Truncates	Ignores	Discards	Conflicts
GASOURCE.TCUSTMER	GASOURCE.TCUSTMER	5	1	0	0	0	0	0	0
GASOURCE.TCUSTORD	GASOURCE.TCUSTORD	5	3	0	2	0	0	0	0

- You can go to GCP Big Query console and check the tables. It may take a short while for tables to be created and loaded.

Figure 8-100 GCPBQ Console

ROW	CUST_CODE	ORDER_DATE	PRODUCT_CODE	ORDER_ID	PRODUCT_PRICE	PRODUCT_AMOUNT	TRANSACTION_ID
1	BLA	1999-01-01 00:00:00 UTC	PRODCA	233	20000.00	10	100
2	BLA	1999-12-31 10:30:00 UTC	ICAB	765	14990.00	1	100
3	BLA	1999-09-09 10:30:00 UTC	ICAB	144	14990.00	1	100

Note

- You can run an initial load with BigQuery replicat. For more information see [BigQuery Handler](#).

8.14 Realtime Message Ingestion to Google Pub/Sub with Oracle GoldenGate for Distributed Applications and Analytics

Overview

This Quickstart covers a step-by-step process showing how to ingest messages to Google Pub/Sub in real-time with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

Google Pub/Sub is a scalable, reliable messaging service that allows asynchronous communication between applications by decoupling message producers (publishers) from message consumers (subscribers).

GG for DAA connects Google Pub/Sub with Google Pub/Sub Handler. GG for DAA reads the source operations from the trail file, formats them, maps to Google Pub/Sub topics and delivers.

- [Prerequisites](#)
- [Install Dependency Files](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.14.1 Prerequisites

- Google Pub/Sub service and topic
- Google Service Account Key

In this Quickstart, a sample trail file `tr` which is shipped with GG for DAA is used. The sample trail file is located at `GG_HOME/opt/AdapterExamples/trail/` in your GG for DAA instance.

8.14.2 Install Dependency Files

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) uses Google Pub/Sub client libraries. You can use the [Dependency Downloader](#) to download the client libraries. Dependency Downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

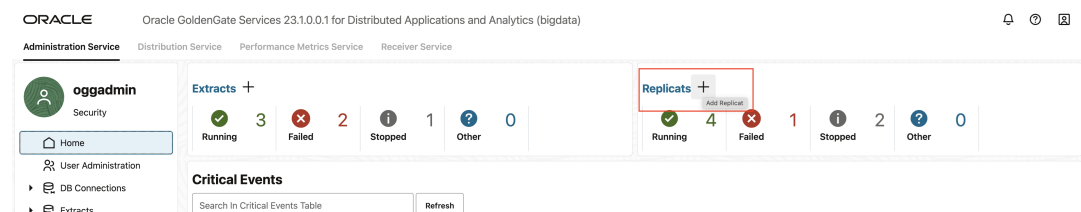
1. In your GG for DAA VM, go to Dependency Downloader utility. It is located at `GG_HOME/opt/DependencyDownloader/`. Locate `googlepubsub.sh`.
2. Run `googlepubsub.sh` with the required version. You can check the version and reported vulnerabilities in [Maven Central](#). This Quickstart uses 1.129.3, which is the latest version when it is published.
3. A new directory `<googlepubsub_version>` is created in `GG_HOME/opt/DependencyDownloader/dependencies`. Note this directory as it will be used in the replicat properties.
 - `/u01/app/ogg/opt/DependencyDownloader/dependencies/googlepubsub_1.129.3`

8.14.3 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

To create a replicat in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA):

1. In the **Administration Service** tab, click the **+** sign to add a replicat.

Figure 8-101 Click + in the Administration Service tab.



2. Select the **Classic Replicat** Replicat Type and click **Next** There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-102 Add Replicat

Add Replicat ✕

Replicat Information Replicat Options Managed Options Parameter File **Properties File**

Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.

Replicat Type

☒ Classic Replicat ☐ Coordinated Replicat

Process Name: GPUBSUB Description:

Next ▶

3. Enter the Replicat options, and click **Next**:
 - a. **Replicat Trail**: Name of the required trail file (if using sample trail, provide as tr)
 - b. **Subdirectory**: Provide as GG_HOME/opt/AdapterExamples/trail/ if using the sample trail.
 - c. **Target**: Google Pub/Sub

Figure 8-103 Replicat Options

Add Replicat ✕

Replicat Information Replicat Options Managed Options Parameter File Properties File

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: Subdirectory: Encryption Profile:

Begin Position in Trail:

Trail Position

Sequence Number: RBA Offset:

Target:

Target:

4. Leave **Managed Options** as is and click **Next**.

Figure 8-104 Managed Options

Add Replicat ✕

Replicat Information Replicat Options **Managed Options** Parameter File Properties File

Use managed options to manage replicat start and auto-start options.

Profile Name: **Default** ▼ Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes: 0 Startup Delay Seconds: 0

Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries: 9 ▼ ▲ Retry Delay Minutes: 0 Retry Delay Seconds: 0

Retries Window Hours: 0 Retries Window Minutes: 0 Retries Window Seconds: 0

◀ Back **Next ▶**

5. Enter **Parameter File** details and click **Next**. In the Parameter File, you can specify source to target mapping or leave it as-is with a wildcard selection.

Figure 8-105 Parameter File

Add Replicat X

Replicat Information ✓
Replicat Options ✓
Managed Options ✓
Parameter File ✓
Properties File 5

Use parameter file to configure replicat parameters and source to target mapping.

```
REPLICAT GPUBSUB
MAP *.* TARGET *.*;
```

Back
Next

6. In Properties File, update the properties marked as **TODO** and click **Create and Run**.

```
# Properties file for Replicat GPUBSUB
#Google Pub/Sub Handler Template
gg.handlerlist=pubsub
gg.handler.pubsub.type=googlepubsub
gg.handler.pubsub.mode=op
#TODO: Set the path to the JSON credentials file
gg.handler.pubsub.credentialsFile=
#TODO: Set the template to resolve the topic name
gg.handler.pubsub.topicMappingTemplate=
#TODO Set the project name
gg.handler.pubsub.projectName=
#TODO: Set the template to resolve the order key - Not required but recommended.
gg.handler.pubsub.orderingKeyMappingTemplate= ${primaryKeys}
gg.handler.pubsub.format=json_row
gg.handler.pubsub.format.metaColumnsTemplate=${objectname[table]},$
{optype[op_type]},${timestamp[op_ts]},${currenttimestamp[current_ts]},$
{position[pos]}
#TODO: Set the path to the Google Pub/Sub client jar files.
gg.classpath=path_to/dependencies/googlepubsub_1.129.3/*
```

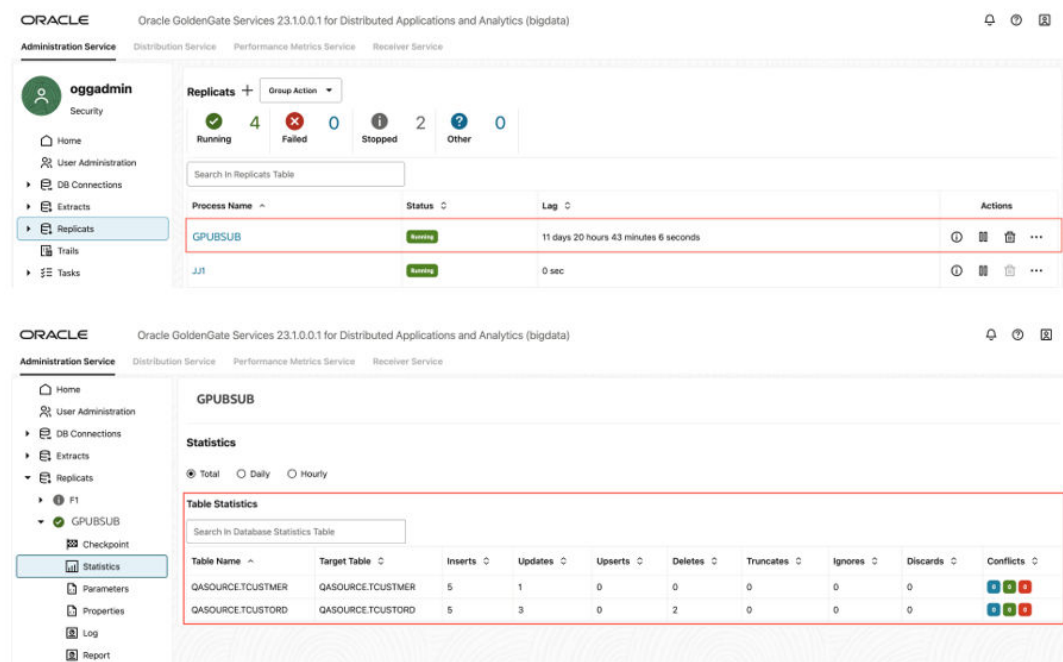
GG for DAA supports dynamic topic mapping by template keywords. For example, if you assign topicMappingTemplate as \${tablename}, GG for DAA will create a topic with the source table name, per each source table and will map the events to these topics.

Oracle recommends using `orderingkeyMappingTemplate=${primaryKeys}`, GG for DAA will send the source operations with the same pk in the same source order. This will guarantee maintaining the order of the source operations while subscriber clients receive the messages.

GG for DAA supports dynamic topic mapping by [Template Keywords](#). For example, if you assign `topicMappingTemplate` as `${tablename}`, GG for DAA will create a topic with the source table name, per each source table and will map the events to these topics. Oracle recommends using `orderingkeyMappingTemplate=${primaryKeys}`, GG for DAA sends the source operations with the same pk in the same source order. This guarantees maintaining the order of the source operations while subscriber clients receive the messages.

7. If replicat starts successfully, it will be in running state. You can go to action/details/statistics to see the replication statistics.

Figure 8-106 Replication Statistics



8. You can go to your [Google Pub/Sub](#) and check the messages

For more information about Google Pub/Sub replication, see [Google Pub/Sub](#).

8.15 Realtime Message Ingestion to Apache Kafka with Oracle GoldenGate for Distributed Applications and Analytics

Overview

This Quickstart covers a step-by-step process showing how to ingest messages to Apache Kafka in real-time with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

Apache Kafka is an open-source platform designed for handling real-time data streams. It allows applications to publish and subscribe to continuous flows of data, making it ideal for building high-performance data pipelines and streaming applications.

GG for DAA connects Apache Kafka with [Kafka Handler](#) and [Kafka Connect Handler](#). GG for DAA reads the source operations from the trail file, formats them, maps to Kafka topics and delivers.

- [Prerequisites](#)
- [Install Dependency Files](#)
- [Create Kafka Producer Properties File](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.15.1 Prerequisites

To successfully complete this Quickstart, you must have the following:

- An Apache Kafka node up and running.

In this Quickstart, a sample trail file (named `tr`) which is shipped with GG for DAA is used. If you want to continue with sample trail file, it is located at `GG_HOME/opt/AdapterExamples/trail/` in your GG for DAA instance.

8.15.2 Install Dependency Files

GG for DAA uses Java SDK provided by Snowflake. You can download the SDKs using [Dependency Downloader](#) utility shipped with GG for DAA. Dependency downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

1. In your GG for DAA VM, go to dependency downloader utility. It is located at `GG_HOME/opt/DependencyDownloader/` and locate `kafka.sh`.
2. Run `kafka.sh` with the required version. You can check the version and reported vulnerabilities in [Maven Central](#). This document uses 3.7.0 which is the latest version when this quick start is published.

Figure 8-107 Run kafka.sh with the required version.



3. A new directory is created in `GG_HOME/opt/DependencyDownloader/dependencies` named as `<kafka_version>`. For example: `/u01/app/ogg/opt/DependencyDownloader/dependencies/kafka_3.7.0`.

8.15.3 Create Kafka Producer Properties File

In GG for DAA instance, create a `producer.properties` file and configure.

For example:

```
bootstrap.servers=localhost:9092
acks = 1
```



```
compression.type = gzip
reconnect.backoff.ms = 1000
```

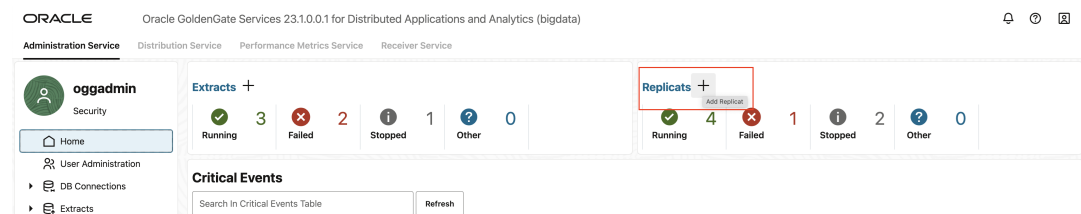
```
value.serializer = org.apache.kafka.common.serialization.ByteArraySerializer
key.serializer = org.apache.kafka.common.serialization.ByteArraySerializer
```

8.15.4 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

To create a replicat in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA):

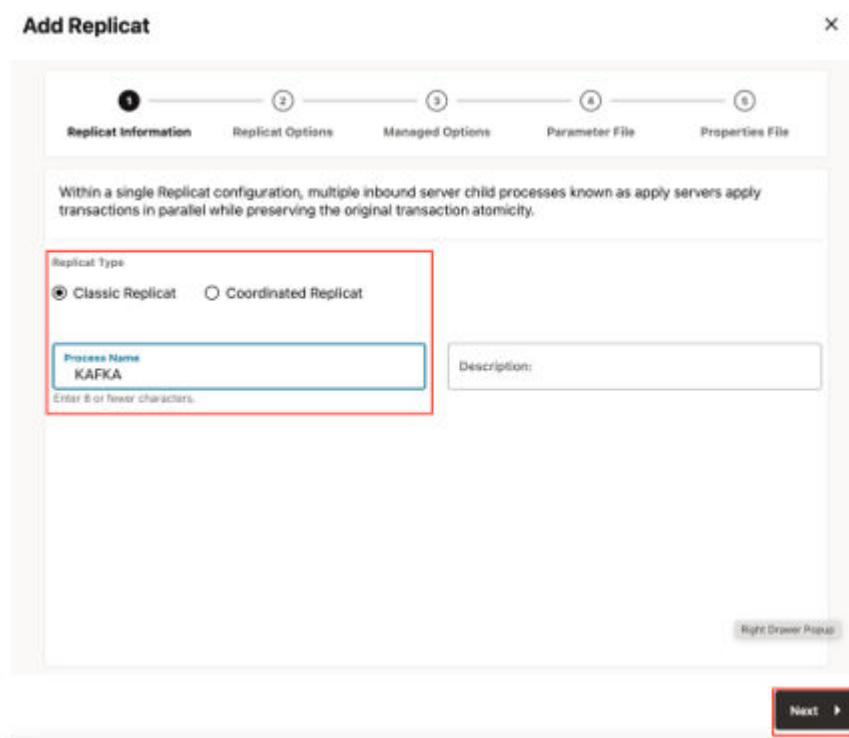
1. In the GG for DAA UI, in the **Administration Service** tab, click the **+** sign to add a replicat.

Figure 8-108 Click + in the Administration Service tab.



2. Select the **Classic Replicat** Replicat Type and click **Next**. There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-109 Add Replicat



3. Enter the basic information, and click **Next**:
 - a. **Target:** Kafka

Figure 8-110 Replicat Options

Add Replicat

Replicat Information **Replicat Options** Managed Options Parameter File Properties File

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: Subdirectory: Encryption Profile:

Begin:

Trail Position

Sequence Number: RBA Offset:

Target:

Target:

Kafka Connect: ☐

4. Leave **Managed Options** as is and click **Next**.

Figure 8-111 Managed Options

Add Replicat [X]

Replicat Information Replicat Options **Managed Options** Parameter File Properties File

Use managed options to manage replicat start and auto-start options.

Profile Name: **Default** Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes: 0 Startup Delay Seconds: 0

Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries: 9 Retry Delay Minutes: 0 Retry Delay Seconds: 0

Retries Window Hours: 0 Retries Window Minutes: 0 Retries Window Seconds: 0

Right Drawer Popup

◀ Back **Next ▶**

5. Enter **Parameter File** details and click **Next**. In the Parameter File, you can specify source to target mapping or leave it as-is with a wildcard selection.

Figure 8-112 Parameter File

Add Replicat

Replicat Information Replicat Options Managed Options **Parameter File** Properties File

Use parameter file to configure replicat parameters and source to target mapping.

```
REPLICAT KAFKA
MAP *.* TARGET *.*;
```

Back Next

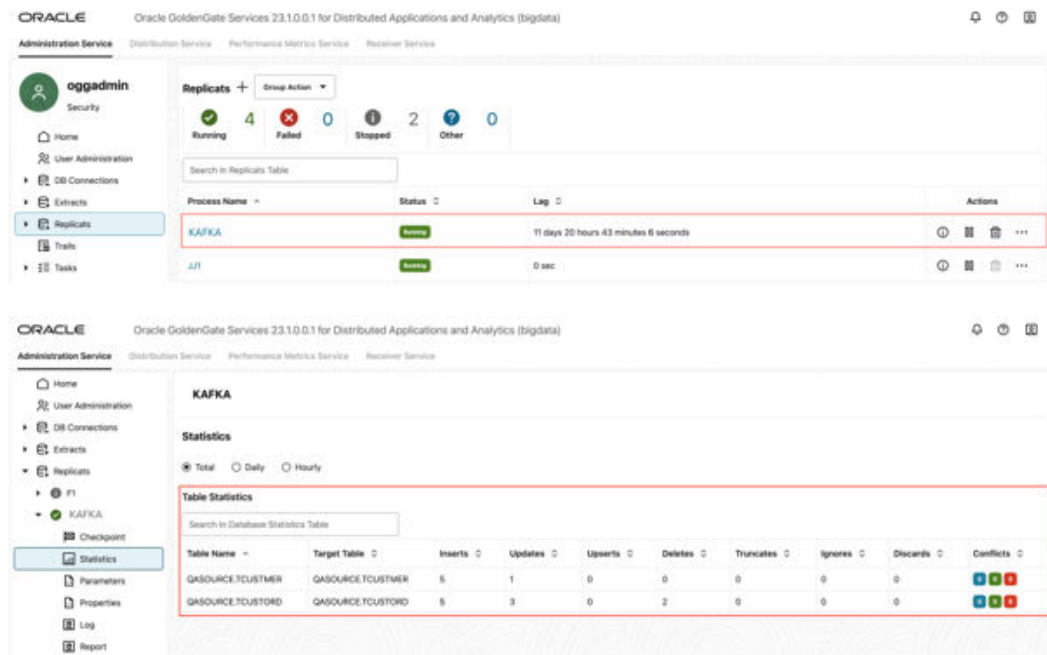
6. In the Properties file, update the properties marked as **TODO** and click **Create and Run**.

```
# Properties file for Replicat KFK
#Kafka Handler Template
gg.handlerlist=kafkahandler
gg.handler.kafkahandler.type=kafka
#TODO: Set the name of the Kafka producer properties file.
gg.handler.kafkahandler.kafkaProducerConfigFile=/path_to/producer.properties
#TODO: Set the template for resolving the topic name.
gg.handler.kafkahandler.topicMappingTemplate=<target_topic_name>
gg.handler.kafkahandler.keyMappingTemplate=${primaryKeys}
gg.handler.kafkahandler.mode=op
gg.handler.kafkahandler.format=json
gg.handler.kafkahandler.format.metaColumnsTemplate=${objectname[table]},${
{optype[op_type]},${timestamp[op_ts]},${currenttimestamp[current_ts]},${
{position[pos]}
#TODO: Set the location of the Kafka client libraries.
gg.classpath= path_to/dependencies/kafka_3.7.0/*
jvm.bootoptions=-Xmx512m -Xms32m
```

GG for DAA supports dynamic topic mapping by [template keywords](#). For example, if you assign `topicMappingTemplate` as `${tablename}`, then GG for DAA creates a topic with the source table name, per each source table and will map the events to these topics. Oracle recommends to use `keyMappingTemplate=${primaryKeys}`, GG for DAA sends the source operations with the same pk to the same partition. This will guarantee maintaining the order of the source operations while delivering to Apache Kafka.

7. If replicat starts successfully, it will be in running state. You can go to action/details/statistics to see the replication statistics.

Figure 8-113 Replicat Statistics



8. You can go to your Kafka topic and check the messages. For more information, see [Apache Kafka](#).

Note

- If target kafka topic does not exist, it will be auto created by GG for DAA if **Auto topic create** is enabled in Kafka cluster. You can use the [template keywords](#) to dynamically assign topic names.
- You can refer to [this blog](#) for improving the performance of the Apache Kafka replication.

8.16 Realtime Data Ingestion into Azure Databricks (unity catalog enabled) with GoldenGate for DAA

Overview

This document covers a step-by-step process showing how to ingest real-time data into Azure Databricks delta tables with GoldenGate for Distributed Applications and Analytics (GG for DAA).

Databricks is a unified, open analytics platform for building, deploying, sharing, and maintaining enterprise-grade data, analytics, and AI solutions at scale.

GG for DAA Databricks handler uses the stage and merge data flow. In stage and merge, the change data is staged in a temporary location in microbatches and eventually merged into to the target table using Merge SQL.

All replication process is automatically handled by [GoldenGate for Distributed Applications and Analytics \(GG for DAA\) Databricks Handler](#)

GG for DAA supports Databricks workspaces configured with and without Databricks Unity Catalog. This Quickstart covers Databricks workspaces with Unity Catalog and creates external tables. External tables are file-backed tables that reference data stored in an external location. Azure Storage container is the external location for Azure Databricks.

- [Prerequisites for Databricks Replication with Unity Catalog](#)
- [Install Dependency Files](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.16.1 Prerequisites for Databricks Replication with Unity Catalog

To successfully complete this Quickstart, you must have the following:

- Databrick workspace with Unity Catalog
- [Storage Credential to Access Azure Storage Account](#)
- [External Location to Access Azure Storage Account](#)
- A schema available in target catalog
- [JDBC URL for Databricks access](#)
- Databricks username/ password (this quick start uses [access tokens](#))

In this Quickstart, a sample trail file (named tr) which is shipped with GG for DAA is used. If you want to continue with sample trail file, it is located at GG_HOME/opt/AdapterExamples/trail/ in your GG for DAA instance.

GG for DAA will create the tables automatically.

8.16.2 Install Dependency Files

GG for DAA uses Java SDK provided by Databrick. You can download the SDKs using [Dependency Downloader](#) utility shipped with GG for DAA. Dependency downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

1. In your GG for DAA VM, go to dependency downloader utility. It is located at GG_HOME/opt/DependencyDownloader/ and locate databricks.sh.
2. Run databricks.sh with the required version. You can check the version and reported vulnerabilities in [Maven Central](#). This document uses 2.6.38 which is the latest version when this Quickstart is published.
3. A new directory is created in GG_HOME/opt/DependencyDownloader/dependencies, which is named as <databricks_jdbc_version>. Make a note of this directory as it will be used in the replicat properties. For example: /u01/app/ogg/opt/DependencyDownloader/dependencies/databricks-jdbc-2.6.36.

Figure 8-114 Run databricks.sh with the required version

```

[oracle@gg4daa-23ai:~/ggdaa23_home/opt/DependencyDownloader]$ ./databricks.sh 2.6.38
openjdk version "11.0.23" 2024-04-16 LTS
Java is installed.
Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08d8cc295161ae)
Maven is accessible.
Root Configuration Script
INFO: This is the Maven binary [../../ggjava/maven-3.9.6/bin/mvn].
INFO: This is the location of the settings.xml file [./docs/settings_np.xml].
INFO: This is the location of the toolchains.xml file [./docs/toolchains.xml].
INFO: The dependencies will be written to the following directory[./dependencies/databricks-jdbc-2.6.38].

```

- Follow the same steps for Azure Storage dependencies. You can run `azure_blob_storage.sh` for downloading Azure Storage dependencies.

Figure 8-115 Run azure_blob_storage.sh for downloading Azure Storage dependencies

```

[oracle@gg4daa-23ai:~/ggdaa23_home/opt/DependencyDownloader]$ ls
aws.sh          cassandra_capture_4x.sh  dependencies  hadoop.sh          hbase_cloudera.sh  kafka_confluent.sh  oracle_nosql_sdk.sh  redis.sh
azure_blob_storage.sh  cassandra_capture_dse.sh  docs          hadoop_azure_cloudera.sh  hbase_hortonworks.sh  kafka_confluent_protobuf.sh  oracle_oci.sh        snowflake.sh
bigquery.sh        cassandra_dse.sh          elasticsearch_java.sh  hadoop_cloudera.sh  internal_scripts    kafka_hortonworks.sh  orc.sh               snowflakestreaming.sh
cassandra.sh        config_proxy.sh           gcs.sh          hadoop_hortonworks.sh  kafka.sh            mongodb.sh           parquet.sh            synapse.sh
cassandra_capture_3x.sh  databricks.sh            googlepubsub.sh  hbase.sh            kafka_cloudera.sh  mongodb_capture.sh   project              velocity.sh
[oracle@gg4daa-23ai:~/ggdaa23_home/opt/DependencyDownloader]$ ./azure_blob_storage.sh 12.13.0

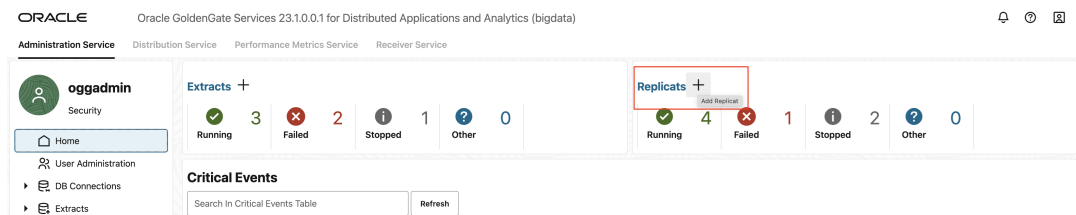
```

A new directory will be created in `GG_HOME/opt/DependencyDownloader/dependencies` named as `<azure-storage-blob_version>`. Make a note of this directory as it will be used in the replicat properties. For example: `/home/oracle/ggdaa23_home/opt/DependencyDownloader/dependencies/azure-storage-blob_12.13.0`

8.16.3 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

To create a replicat in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA):

- In the GG for DAA UI, in the **Administration Service** tab, click the **+** sign to add a replicat.

Figure 8-116 Click + in the Administration Service tab.

- Select the **Classic Replicat** Replicat Type and click **Next**. There are two different Replicat types available: Classic and Coordinated. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel.

Figure 8-117 Select a Replicat Option

Add Replicat ✕

1 — 2 — 3 — 4 — 5
Replicat Information Replicat Options Managed Options Parameter File Properties File

Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.

Replicat Type

☒ Classic Replicat ☐ Coordinated Replicat

Process Name: DBRDELTA Description:

Next ➤

3. Enter the Replicat options and click **Next**:
 - a. **Trail Name**: Name of the required trail file (if using sample trail, provide as tr)
 - b. **Subdirectory**: Enter GG_HOME/opt/AdapterExamples/trail/ if using the sample trail.
 - c. **Target**: Databricks
 - d. **Available Staging Locations**: Azure Data Lake Storage

Figure 8-118 Provide Replicat Options and Select Target

Add Replicat ✕

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name tr	Subdirectory /AdapterExamples/trail/	Encryption Profile LocalWallet

Begin
Position in Trail

Trail Position

Sequence Number 0	RBA Offset 0

Target:

Target:
Databricks

☒ Stage and Merge - High volume ingestion for mixed workloads

Available staging locations

Available staging locations
Azure Data Lake Storage

◀ Back Next ▶

4. Leave **Managed Options** as is and click **Next**.

Figure 8-119 Managed Options

Add Replicat ✕

Replicat Information Replicat Options **Managed Options** Parameter File Properties File

Use managed options to manage replicat start and auto-start options.

Profile Name: Default ▼ Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes Startup Delay Seconds

Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries Retry Delay Minutes Retry Delay Seconds

Retries Window Hours Retries Window Minutes Retries Window Seconds

◀ Back Next ▶

5. Enter **Parameter File** details and click **Next**. In the Parameter File, you can specify source to target mapping. If you're using the sample trail file (tr) provide as followed: MAP QASOURCE.*, TARGET <your_schema_name>.*;

Figure 8-120 Parameter File

Add Replicat

Replicat Information Replicat Options Managed Options Parameter File Properties File

Use parameter file to configure replicat parameters and source to target mapping.

```
REPLICAT DBRDELTA
MAP QASOURCE.TCUSTMER, TARGET QATARGET.QUICK;
```

Back Next

6. In the Properties file, update the properties marked as **TODO** and click **Create and Run**.

```
# Properties file for Replicat DBRDELTA
# Configuration to load GoldenGate trail operation records into Databricks using
ADLS Gen2 stage.
# Note: Recommended to only edit the configuration marked as TODO
gg.target=databricks
gg.stage=abs
# Azure Blob Event handler.
#TODO: Edit Azure connection settings
gg.eventhandler.abs.bucketMappingTemplate=<azure_adls_gen2_container_name>
gg.eventhandler.abs.accountName=<azure_storage_account_name>
gg.eventhandler.abs.accountKey=<azure_storage_account_key>

# Databricks Event Handler.
```

Edit your Databricks JDBC:

```
#TODO: Edit JDBC ConnectionUrl
gg.eventhandler.databricks.connectionURL=jdbc:databricks://<server-
hostname>:443;httpPath=<http-
path>[;<setting1>=<value1>;<setting2>=<value2>;<settingN>=<valueN>];EnableArrow=0
#TODO: Edit JDBC username or 'token'
gg.eventhandler.databricks.UserName=token
#TODO: Edit JDBC password
gg.eventhandler.databricks.Password=<password>
```

Provide path to dependency jar files that you downloaded in prerequisites

#TODO: Edit the classpath to include Azure Blob Event Handler dependencies and Databricks JDBC driver.

gg.classpath=/path/to/abs_dependencies/*: path/to/databricks_dependencies/*

#TODO: Provide sufficient memory (at least 8GB).

jvm.bootoptions=-Xmx8g -Xms1g

Figure 8-121 Properties File

Add Replicat ✕

Replicat Information
Replicat Options
Managed Options
Parameter File
5
Properties File

Configure replicat properties marked as #TODO.

```
# Properties file for Replicat DBRDELTA
# Configuration to load GoldenGate trail operation records into Databricks using ADLS Gen2 stage.
# Note: Recommended to only edit the configuration marked as TODO
gg.target=databricks
gg.stage=abs
# Azure Blob Event handler.
gg.eventhandler.abs.bucketMappingTemplate=databricks-external
gg.eventhandler.abs.accountName=ocigqtest
gg.eventhandler.abs.accountKey=ORACLEWALLETPASSWORD[ADLS OracleGoldenGate]

# Databricks Event Handler.
#TODO: Edit JDBC ConnectionUrl
gg.eventhandler.databricks.connectionURL=jdbc:databricks://adb-3393173566748244.4.azuredatabricks.net:443/
default;transportMode=http;ssl=1;httpPath=sql/protocolv1/o/3393173566748244/0723-113428-mssk6wvh;
#TODO: Edit JDBC username or 'token'
gg.eventhandler.databricks.Username=ORACLEWALLETUSERNAME[Databricks OracleGoldenGate]
#TODO: Edit JDBC password
gg.eventhandler.databricks.Password=ORACLEWALLETPASSWORD[Databricks OracleGoldenGate]

#TODO: Edit the classpath to include Azure Blob Event Handler dependencies and Databricks JDBC driver.
gg.classpath=/home/oracle/ggdaa23_home/opt/DependencyDownloader/dependencies/azure-storage-blob_12.13.0/
*/home/oracle/ggdaa23_home/opt/DependencyDownloader/dependencies/databricks-jdbc-2.6.36/*
#TODO: Provide sufficient memory (at least 8GB).
jvm.bootoptions=-Xmx8g -Xms1g
EnableArrow=0
```

← Back
Create
Create and Run

See [Databricks](#) for more replicat configuration details.

When replicat starts successfully, it will be in running state. Note that by default, batch window is set to 3 mins. It may take a short moment for your data to be loaded to Databricks. You can go to action/details/statistics to see the replication statistics.

Figure 8-122 Replication Statistics

The first screenshot shows the 'Replicates' section in the Oracle GoldenGate Services console. It displays a summary of replication status: 1 Running, 0 Failed, 0 Stopped, and 0 Other. Below this, a table lists the replicates:

Process Name	Status	Lag	Actions
DBRDELTA	Running	0 sec	[Stop] [Refresh] [More]

The second screenshot shows the 'DBRDELTA (NONINTEGRATED)' statistics page. It displays 'Table Statistics' for the QASOURCE.CUSTOMER table. The statistics are as follows:

Table Name	Target Table	Inserts	Updates	Upserts	Deletes	Truncates	Ignores	Discards	Conflicts
QASOURCE.CUSTOMER	QATARGET.QUICK	5	1	0	0	0	0	0	0

- You can go to Databricks and check the tables. It may take a short moment for tables to be created and loaded.

Figure 8-123 Databricks Tables

The screenshot shows the Databricks console with a successful SQL query execution. The query is:

```
USE CATALOG ucenabedcatalog;
SELECT * FROM qatarget.quick;
```

The result is a table with 5 rows and 4 columns: CUST_CODE, NAME, CITY, and STATE. The data is as follows:

CUST_CODE	NAME	CITY	STATE
1	ANN'S BOATS	NEW YORK	NY
2	BO SOFTWARE CO.	SEATTLE	WA
3	ROCKY FLYER INC.	DENVER	CO
4	DAVE'S PLANES INC.	TALLAHASSEE	FL
5	BILL'S USED CARS	DENVER	CO

The table has 5 rows and 4.05 seconds runtime. The result is stored as a Spark DataFrame and can be used in other Python cells.

Note

For more information, see [Databricks](#).

8.17 Realtime Data Ingestion into Oracle AI Data Platform with Oracle GoldenGate for DAA

Overview

This Quickstart covers a step-by-step process showing how to ingest parquet files into Oracle AI Data Platform (AIDP) in real-time with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

Oracle AI Data Platform is a managed service that unifies data lake, catalog, compute, and workflow orchestration into a single service. It allows the users to securely discover, prepare, and govern structured and unstructured data, while enabling large-scale analytics and AI/ML workloads with Apache Spark for building data-driven applications and accelerating business insights.

GG for DAA AIDP handler uses the stage and merge data flow. In stage and merge, the change data is staged in an OCI Object Storage bucket in microbatches and eventually merged into the target delta tables managed by AIDP. All replication process is automatically handled by [Oracle AI Data Platform](#).

- [Prerequisites](#)
- [Install Required Dependency Files](#)
- [Configure Credentials for Oracle Cloud Infrastructure](#)
- [Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics](#)

8.17.1 Prerequisites

To successfully complete this Quickstart, you must have the following:

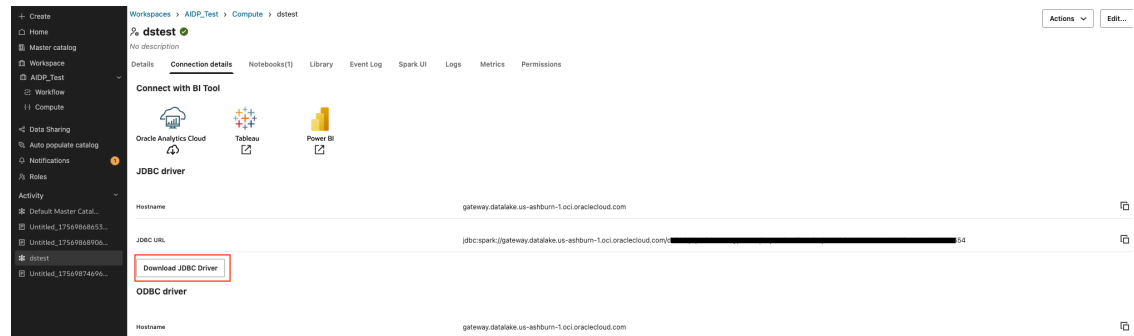
- Oracle Cloud Infrastructure account set up for Oracle AI Data Platform.
- Simba JDBC driver for Apache Spark. You can download the Simba JDBC driver from the cluster detail page of the Oracle AI Data Platform console.

In this Quickstart, a sample trail file (named `tr`) which is shipped with GG for DAA is used. If you want to continue with sample trail file, it is located at `GG_HOME/opt/AdapterExamples/trail/` in your GG for DAA instance.

8.17.2 Install Required Dependency Files

GG for DAA uses client libraries in the replication process and these libraries need to be downloaded before setting up the replication process. You can use [dependency downloader](#) to download the client libraries. Dependency Downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

GG for DAA uses Simba JDBC driver that you can download from AIDP console. You can download the Simba JDBC driver from Oracle AI Data Platform Workspace / Compute / Connection details.

Figure 8-124 JDBC Driver

The required dependency files for OCI Object Storage can be downloaded using Dependency Downloader utility available in GG for DAA. The Dependency Downloader is a set of shell scripts that downloads dependency jar files from Maven and other repositories.

1. In your GG for DAA VM, go to *Dependency Downloader* utility located at `GG_HOME/opt/DependencyDownloader/`.
2. Run the `oracle_oci.sh` with the required version.

Figure 8-125 Run oracle_oci.sh with the required version

```
[opc@ggd4a239 ~]$ ls -la /u01/app/ogg/opt/DependencyDownloader/
total 12
drwxr-xr-x 2 root root 4096 Sep 11 10:10 .
drwxr-xr-x 1 root root 4096 Sep 11 10:10 ..
-rwxr-xr-x 1 root root 1024 Sep 11 10:10 oracle_oci_3.2.0.sh
```

3. A new directory is created in `GG_HOME/opt/DependencyDownloader/dependencies`. For example, `/u01/app/ogg/opt/DependencyDownloader/dependencies/oracle_oci_3.2.0/*`. Take a note of this directory.

8.17.3 Configure Credentials for Oracle Cloud Infrastructure

You need to create a configuration file to authenticate into OCI. The ideal configuration file include user, fingerprint, key_file, tenancy, and region with their respective values. The default configuration file name and location is `~/oci/config`. For more information, see [required keys and ocids](#) document.

Sample Configuration File

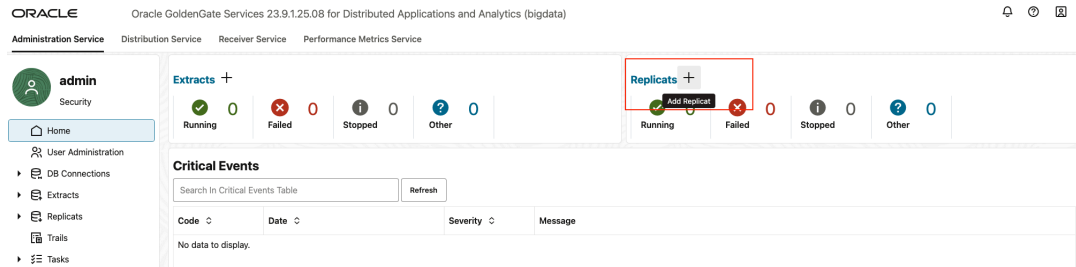
```
[DEFAULT]
user=ocid1.user.oc1..mockValue
fingerprint=mockFingerPrintValue
tenancy=ocid1.compartment.oc1..mockValue
region=us-phoenix-1
key_file=<path to your private keyfile>
```

8.17.4 Create a Replicat in Oracle GoldenGate for Distributed Applications and Analytics

To create a replicat in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA):

1. Go to **Administration Service** and click the **+** sign to add a replicat.

Figure 8-126 Click + in the Administration Service tab



2. Select the **Replicat Type** and click **Next**.
There are two different Replicat types available: **Classic Replicat** and **Coordinated Replicat**. Classic Replicat is a single threaded process whereas Coordinated Replicat is a multithreaded one that applies transactions in parallel. Coordinated Replicat results in multiple files being created as there is a multithreaded process running.

Figure 8-127 Select a Replicat Option

Add Replicat

1 Replicat Information 2 Replicat Options 3 Managed Options 4 Parameter File 5 Properties File

Within a single Replicat configuration, multiple inbound server child processes known as apply servers apply transactions in parallel while preserving the original transaction atomicity.

Replicat Type

☒ Classic Replicat ☐ Coordinated Replicat

Process Name
AIDP
Enter 8 or fewer characters.

Description:

Next

3. Enter the **Replicat Options** and click **Next**:

- a. **Replicat Trail:** Name of the required trail file. For sample trail, provide tr.
- b. **Subdirectory:** Enter GG_HOME/opt/AdapterExamples/trail/ if using the sample trail.
- c. **Target:** Oracle AI Data Platform

Figure 8-128 Provide Replicat Options and Select Target

Add Replicat

1 **Replicat Information** 2 **Replicat Options** 3 **Managed Options** 4 **Parameter File** 5 **Properties File**

Provide replicat options and select target. Use encryption profile to configure information that is used to retrieve a masterkey from a KMS.

Replicat Trail

Name: tr

Subdirectory:

Encryption Profile: LocalWallet

Begin Position in Trail:

Trail Position

Sequence Number: 0

RBA Offset: 0

Target

Target: Oracle AI Data Platform

☒ Stage and Merge - High volume ingestion for mixed workloads

Staging location

Staging location: OCI Object Storage

Back Next

- d. Leave **Managed Options** as is and click **Next**.

Figure 8-129 Managed Options

Add Replicat

Replicat Information Replicat Options **Managed Options** Parameter File Properties File

Use managed options to manage replicat start and auto-start options.

Profile Name: **Default** Critical to deployment health ☐

Auto Start ☐ Startup Delay Minutes: 0 Startup Delay Seconds: 0

Auto Restart ☐ Restart on Failure only ☐ Disable Task After Retries ☐

Max Retries: 9 Retry Delay Minutes: 0 Retry Delay Seconds: 0

Retries Window Hours: 0 Retries Window Minutes: 0 Retries Window Seconds: 0

[Back](#) [Next](#)

- e. Enter **Parameter File** details and click **Next**.
In the **Parameter File**, you can either specify source to target mapping or leave it as-is with a wildcard selection. If **Coordinated Replicat** is selected as the Replicat Type, an additional parameter needs to be provided:

```
TARGETDB LIBFILE libggjava.so SET property=<ggbd-deployment_home>/etc/
conf/ogg/your_replicat_name.properties
```

Figure 8-130 Parameter File

Add Replicat

Replicat Information Replicat Options Managed Options **Parameter File** Properties File

Use parameter file to configure replicat parameters and source to target mapping.

```
REPLICAT AIDP
MAP *.* , TARGET *.*;|
```

Back Next

- f. In the **Properties File**, update the properties marked as **TODO** and click **Create and Run**.

Note

Before clicking **Create and Run** you need to copy and paste the provided *property list* into **Properties File**, update as required and click **Create and Run**.

```
# Properties file for Replicat
AIDP# Configuration to load
GoldenGate trail operation records into AI Data Platform using OCI
object store staging
location.# Note: Recommended to only edit
the configuration marked as TODO

gg.target=aidp
gg.stage=oci
# The OCI Event handler
```

```
#TODO: Edit the OCI Config file path
gg.eventhandler.oci.configFilePath=/path_to/.oci/config
#TODO: Edit the OCI profile
gg.eventhandler.oci.profile=<your_oci_profile_name>
#TODO: Edit the OCI region
gg.eventhandler.oci.region=<your_oci_region>
#TODO: Edit the OCI compartment OCID
gg.eventhandler.oci.compartmentID=<your_compartment_ocid, eg;
ocid1.compartment.oc1..aaaaaaaaftrzllvla63f5von...>
#TODO: Edit the OCI bucket name
gg.eventhandler.oci.bucketMappingTemplate=<your_bucket_name>

# Oracle AI Data Platform Event Handler.
#TODO: Edit JDBC ConnectionUrl
gg.eventhandler.aidp.connectionURL=<your_aidp_jdbc, eg; jdbc:spark://
gateway.datalake.us-ashburn-1.oci.oraclecloud.com/
default;SparkServerType=IDL;httpPath=cliservice/
393dcb48-302...;OCIProfile=<your_oci_profile_name>;
#TODO: Edit the classpath to include OCI Event handler dependencies and
Simba JDBC driver.
gg.classpath=/home/oracle/dependencies/*:/home/oracle/install/gg/opt/
DependencyDownloader/dependencies/oracle_oci_3.0.0/*
```

4. If replicat starts successfully, then it is in running state. Go to action/details/statistics to see the replication statistics.

Figure 8-131 Replication Statistics

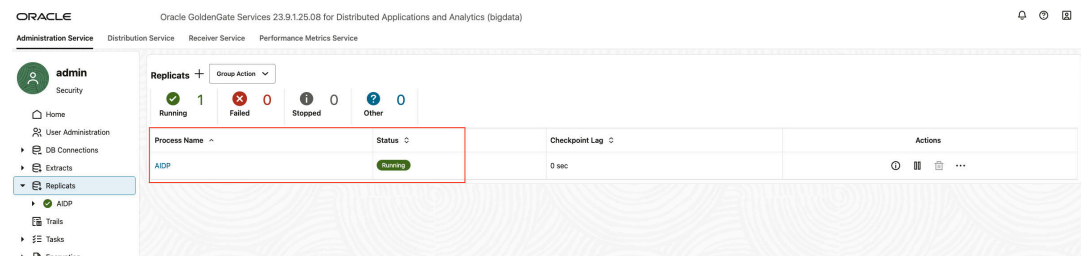
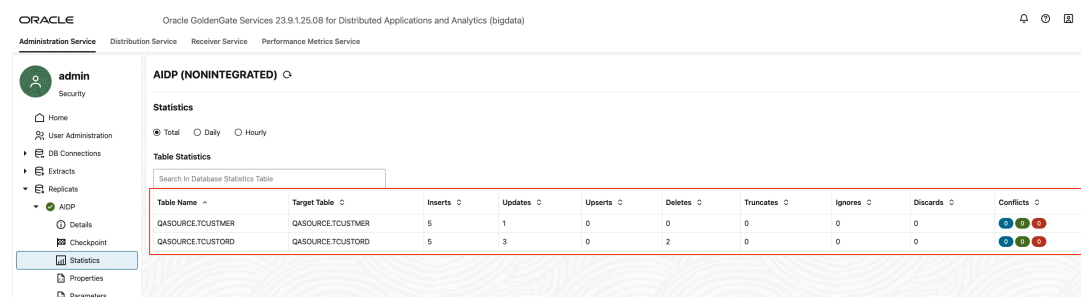
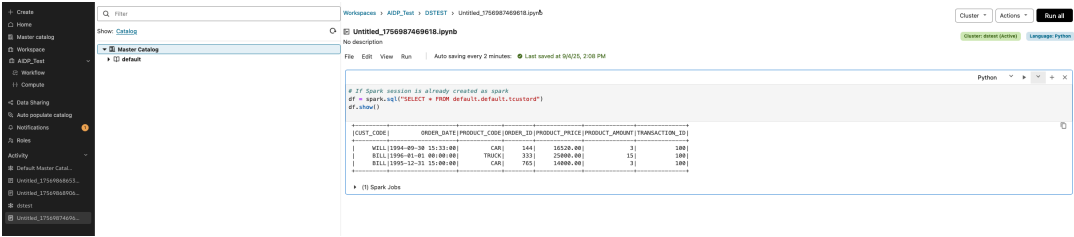


Figure 8-132 Replication Statistics



5. Go to AI Data Platform console and check the tables. It may take a short moment for tables to be created and loaded.

Figure 8-133 AI Data Platform



9

Best Practices

This chapter covers recommended best practices for enhancing performance.

- [Apache Kafka](#)
The Kafka Handler is designed to stream change data from an Oracle GoldenGate trail to a Kafka topic.
- [Cloud Object Storage Replication](#)
- [Snowflake](#)
The Snowflake Replicat delivers changed data from Oracle GoldenGate trail files into Snowflake target tables.

9.1 Apache Kafka

The Kafka Handler is designed to stream change data from an Oracle GoldenGate trail to a Kafka topic.

This chapter describes Oracle GoldenGate for Apache Kafka Replication Best Practices.

- [Configuration Considerations and Best Practices](#)
- [Oracle GoldenGate Kafka Replicat Topic Mapping](#)
- [Oracle GoldenGate Kafka Replicat Schema Registry and Schema Propagation](#)
- [Oracle GoldenGate Kafka Replicat Partition Selection and Order Consistency](#)
- [Propagating Additional Metadata with Oracle GoldenGate Kafka Replicat](#)
- [Oracle GoldenGate Kafka Replicat Message Modes and Maintaining Transactional Boundaries](#)
Oracle GoldenGate Kafka Replicat supports two different message modes:
- [Oracle GoldenGate Kafka Replicat Message Formatting](#)
- [Security Considerations and Best Practices](#)
- [Performance Considerations](#)
To improve overall performance, you should review your entire data pipeline, including the Extract processes.

9.1.1 Configuration Considerations and Best Practices

Oracle GoldenGate's Replicat for Apache Kafka is configured using three core elements:

- **Configuration Properties:** Configuration properties for the Oracle GoldenGate Replicat for Apache Kafka are key-value settings that control how data is formatted, mapped, and delivered from GoldenGate to Apache Kafka topics. These properties control message format, topic mapping, schema propagation, and other integration behaviors to meet data-streaming requirements.
- **Message Formatters:** Oracle GoldenGate [Pluggable Formatters](#) are modular formatting components that convert change-data operations captured in the GoldenGate trail files into structured messages for Kafka targets. Formatters support multiple serialization formats

(JSON, Avro, and others) and allow fine-grained control over how messages are represented in Kafka topics.

- **Kafka Producer Properties:** These are standard properties defined by Kafka that control the behavior of the Kafka Producer client. When configuring Oracle GoldenGate Replicat for Apache Kafka, these properties are passed through to the underlying Kafka producer to govern aspects such as broker connectivity, security, message delivery reliability, and throughput.

These three elements should be designed and tuned together to ensure optimal performance, consistency, and maintainability of Oracle GoldenGate replication for Apache Kafka.

9.1.2 Oracle GoldenGate Kafka Replicat Topic Mapping

- [Table-per-Topic Pattern](#)
The Table-per-Topic is a strategic Kafka design in which each source database table maps to its own dedicated Kafka topic. All CDC events from a given table are published exclusively to that topic. Oracle recommends Table-per-Topic pattern as it enhances downstream processing, simplifies troubleshooting, and aligns with strong schema management practices.
- [All-Tables-to-Single-Topic Pattern](#)
The All-Tables-to-Single-Topic Pattern (also called the monotopic pattern) consolidates CDC events from multiple source database tables into a single Kafka topic.

9.1.2.1 Table-per-Topic Pattern

The Table-per-Topic is a strategic Kafka design in which each source database table maps to its own dedicated Kafka topic. All CDC events from a given table are published exclusively to that topic. Oracle recommends Table-per-Topic pattern as it enhances downstream processing, simplifies troubleshooting, and aligns with strong schema management practices.

Implementation and Configuration

Oracle GoldenGate Kafka Replicat controls topic assignment at runtime using the *topicMappingTemplate* property in [Setting Up and Running the Kafka Connect Handler](#). This configuration supports both static topic assignments and dynamic mapping via [Template Keywords](#) for example `${tableName}`.

To implement the Table-per-Topic pattern, set the property as follows:

```
gg.handler.kafkaconnect.topicMappingTemplate=${tableName}
```

This dynamically assigns each table's data to a distinct Kafka topic named after the source table.

Considerations and Best Practices

- **Topic Creation and Configuration:**
If the Kafka broker property `auto.create.topics.enable` is set to `true` in `server.properties`, Oracle GoldenGate can automatically create target topics.

Recommendation: For production environments, it is best practice to pre-create topics with appropriate configurations (specifically replication factor, partition count, and retention policy). This ensures performance, high availability, and data lifecycle management are explicitly controlled. Disabling auto topic creation after pre-creation is a typical operational best practice.

- **Schema Registry Integration:**

When using Oracle GoldenGate Kafka Replicat with a schema registry (such as Confluent Schema Registry) and an Avro formatter, the Table-per-Topic pattern is strongly recommended. Each source table is naturally associated with a unique Avro schema in the registry, simplifying schema evolution and enforcing data-contract compliance.

Example:

The source table **HR.EMPLOYEES** mapped to a Kafka topic named **EMPLOYEES** will, by default, result in the registration of its Avro schema under the subject name: `hr.employees-value`.

9.1.2.2 All-Tables-to-Single-Topic Pattern

The All-Tables-to-Single-Topic Pattern (also called the monotopic pattern) consolidates CDC events from multiple source database tables into a single Kafka topic.

This approach significantly simplifies Kafka topic management and reduces the number of topics that must be created, monitored, and configured. However, this approach transfers the complexity to the consumer side, requiring downstream applications must inspect message metadata (such as schema name and table name) to filter and route records appropriately. The Oracle GoldenGate Replicate for Kafka supports the injection of metadata fields (such as table name and transaction id) into the message header, which reduces the complexity of downstream consumer applications.

When used with a schema registry, placing multiple disparate schemas into the same Kafka topic significantly complicates schema management. Although schema registry can technically support this pattern, it is not considered a best practice, as it often requires relaxing schema compatibility settings to accommodate heterogeneous message structures. This introduces additional complexity for schema evolution and increases the burden on consumers, which must implement flexible validation and handling logic for multiple schema versions.

Furthermore, because all events are routed to a single topic, consumer parallelism is inherently limited by the number of partitions on that one topic, which can create a bottleneck for high-throughput, multi-table workloads.

Implementation and Configuration

To implement the all-tables-to-single-topic pattern, set the topic mapping property as follows:

```
gg.handler.kafka.topicMappingTemplate=<target_topic_name>
```

Considerations and Best Practices

Ensure that each message published to the topic includes metadata fields (such as schema name and table name) to facilitate correct identification and downstream processing.

Using the `metaHeadersTemplate` property, you can include [Template Keywords](#) (for example, `${tableName}`) in Kafka message headers. This allows the downstream consumer applications to easily read the key header metadata, thereby reducing the amount of parsing and processing needed in the consumer. For example:

```
gg.handler.kafka.metaHeadersTemplate=${tableName}
```


9.1.3 Oracle GoldenGate Kafka Replicat Schema Registry and Schema Propagation

Oracle GoldenGate Replicate for Kafka supports integration with Confluent Schema Registry when using Avro, Protobuf, or JSON converters. When Oracle GoldenGate publishes messages, it automatically registers the corresponding schemas in the schema registry, enabling consumer applications to retrieve and apply them during deserialization. When a change occurs in the source schema, Oracle GoldenGate Kafka Replicat registers the updated schema as a new version in the Schema Registry.

Implementation and Configuration

Schema registry connectivity and authentication can be configured through the Kafka producer .properties file. Oracle GoldenGate supports Basic Authentication, TLS, and, mTLS for secure communication with the Confluent Schema Registry. Sample configuration files are provided in the [Kafka Connect Handler Performance Considerations](#) in the Oracle GoldenGate documentation.

Considerations and Best Practices

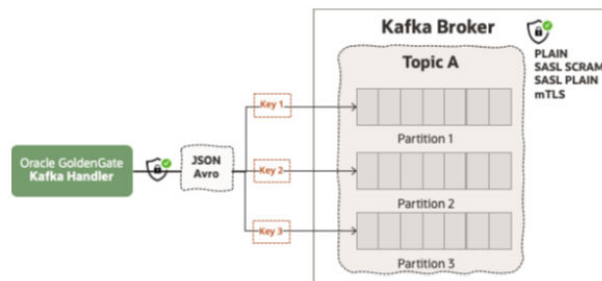
- The Table-per-Topic pattern is recommended when using a schema registry. Each source table is naturally associated with a unique Avro schema in the registry, simplifying schema evolution and ensuring compliance with data-contract standards.
- The Confluent Schema Registry enforces configurable compatibility modes to guarantee that newly registered schema versions remain compatible with previous ones. This is particularly important when Oracle GoldenGate Replicate for Kafka automatically registers new schema versions after source-side DDL changes, for example, an ALTER TABLE statement.
- Backward and Full compatibility modes are the most suitable options for Oracle GoldenGate Replication for Kafka, as they ensure seamless schema evolution—allowing new schema versions to remain readable by consumers using older schemas, and thereby maintain uninterrupted data streaming and stable deserialization across CDC pipelines.

9.1.4 Oracle GoldenGate Kafka Replicat Partition Selection and Order Consistency

Partition selection is key to ensuring both balanced throughput and data consistency in Oracle GoldenGate Replicate for Kafka pipeline. The method Oracle GoldenGate uses to determine which Kafka partition each event is written to directly influences message ordering, transactional integrity, and downstream processing behavior.

Implementation and Configuration

In Oracle GoldenGate replicate for Kafka, message partitioning is governed by Kafka's native partitioning logic, which relies on the message key assigned at runtime. The keyMappingTemplate property in Oracle GoldenGate defines how the message key is constructed, influencing Kafka's partition assignment. If the configuration produces a static key, all messages are routed to the same partition; if the key changes infrequently, partition assignments also shifts minimally. For optimal performance, the keyMappingTemplate should be configured to ensure an even key distribution among partitions, while also maintaining source-order consistency for individual logical records relating to the same row.

Figure 9-1 Oracle GoldenGate Kafka Replicate Partition Selection

Considerations and Best Practices

Oracle recommends using source primary keys for partition selection, if table per topic pattern is implemented. This can be achieved by using `${primaryKeys}` [Template Keywords](#) with `keyMappingTemplate` property.

For example:

```
gg.handler.kafkaconnect.keyMappingTemplate=${primaryKeys}
```

When source primary keys are used as the Kafka message key, operations for the same source primary key are sent to the same target partition, because the Kafka producer uses the hash of the key to calculate the partition ID. In this way, throughput is balanced as messages are evenly distributed between partitions and source order is maintained for the operations having the same message key value.

In some cases, customers may have large transactions spread over to multiple tables. For such cases, Oracle recommends using all-tables-to-single topic pattern and assigning `${xid}` (source transaction id) as the message key.

For example:

```
gg.handler.kafkaconnect.keyMappingTemplate=${xid}
```

This ensures that all source operations from the same transaction (XID) are written to the same partition, guaranteeing that the entire transaction's life cycle is processed in sequential source order by a single consumer thread.

9.1.5 Propagating Additional Metadata with Oracle GoldenGate Kafka Replicat

The need to enrich CDC events with contextual information requires injecting additional metadata fields into the Kafka messages. The Oracle GoldenGate Kafka Replicat can inject this context-based metadata at runtime.

Implementation and Configuration

Oracle GoldenGate Replicat for Kafka leverages the `metaColumnsTemplate` property and a rich library of [Metacolumn Keywords](#) to inject contextual key-value pairs into the Kafka messages. This critical best practice separates operational context (such as source table name, timestamp, and operation type) from the core data payload. This separation ensures the

core data schema remains clean and simple, while still providing necessary metadata for downstream consumers to audit, filter, or route events.

For example, to inject the source table name and the operation type into the payload, you would set the property as follows:

```
gg.handler.kafkaconnect.metaColumnsTemplate=${objectname},${optype}
```

Considerations and Best Practices

To avoid conflicts with default header names or to improve clarity, you can define a custom name for the injected field using the square bracket syntax. If a custom name is not provided, the default metacolumn name is used.

For example:

```
gg.handler.kafkahandler.format.metaColumnsTemplate=${objectname[table]},${optype}
```

In this example, `${objectname}` is injected with the field name `table` and `optype` is injected with default field name.

9.1.6 Oracle GoldenGate Kafka Replicat Message Modes and Maintaining Transactional Boundaries

Oracle GoldenGate Kafka Replicat supports two different message modes:

- [Operation Mode \(OP\)](#)
- [Transaction Mode \(TX\)](#)

9.1.6.1 Operation Mode (OP)

In Operation Mode (OP) mode, Oracle GoldenGate Kafka Replicat processes each source record (insert, update, delete) as an individual operation and produces a corresponding Kafka record for each. This establishes a one-to-one relationship between the number of incoming source operations and the number of Kafka messages produced, resulting in a high-frequency, non-batched delivery mechanism.

Performance in Operation Mode mode can be optimized using the `GROUPTRANSOPS` parameter. This parameter controls how many messages are grouped and flushed together, improving throughput by reducing the number of write operations to Kafka. For more details, see [Kafka Connect Handler Performance Considerations](#).

9.1.6.2 Transaction Mode (TX)

In Transaction Mode (TX) mode, Oracle GoldenGate Kafka Replicat treats each source database transaction as a single, atomic Kafka record. All operations within a transaction are concatenated and serialized into the value of a single Kafka record. This means that each Kafka message can represent one or multiple database operations (up to the total count within the transaction), preserving transactional atomicity. TX mode is particularly useful where maintaining transactional boundaries and atomic delivery in Kafka is required.

Implementation and Configuration

Oracle GoldenGate Kafka Replicat leverages the [Template Keywords](#) mode property to assign the message mode.

For example:

```
gg.handler.kafkaconnect.mode=op
```

Considerations and Best Practices

Operation Mode (OP)

Consider increasing the GROUPTRANSOPS parameter (default: 1,000) to improve throughput by batching more records prior to flushing. Values up to 20,000 can be configured, but optimal settings may vary based on workload and infrastructure.

When using the table-per-topic pattern, it is recommended to map the Kafka message key to the source operation's primary key.

For example:

```
gg.handler.kafkaconnect.keyMappingTemplate=${primaryKeys}
```

Transaction Mode (TX)

- TX mode groups all source operations within the same transaction into a single Kafka message, ensuring transactional atomicity. However, this may result in very large messages, potentially exceeding the maximum allowed message size in Kafka.
- Because TX mode inherently groups operations, the impact of GROUPTRANSOPS is lessened. However, if the goal is absolute transactional atomicity, the GROUPTRANSOPS setting should be reviewed, although keeping the default or a high value may still be necessary to optimize the flush frequency.
- For handling rare, large source transactions-often due to batch processing-use the MAXTRANSOPS parameter to split these transactions into smaller Kafka messages. This way, you can handle very large source transactions while replicating to Kafka.
- To ensure all messages from a transaction are written to the same Kafka partition, configure the key mapping to use the source transaction ID by using keyMappingTemplate property.
- For example:

```
gg.handler.kafkaconnect.keyMappingTemplate=${xid}
```

9.1.7 Oracle GoldenGate Kafka Replicat Message Formatting

The Oracle GoldenGate Kafka Replicat offers a comprehensive suite of message formatting options to align with diverse streaming and consumption requirements. JSON and Avro are the most preferred formats for high-volume Kafka replication, with support also extending to the CloudEvents Format.

For both Avro and JSON, the Oracle GoldenGate Kafka Replicat supports following two primary formatting methodologies:

- Operation-Based Formatting

- Row-Based Formatting

The key difference between these two methods is how they include a row's before-and-after images in the message payload for each source DML operation (INSERT, UPDATE, DELETE) and for TRUNCATE.

Table 9-1 Operation vs Row Formatting

Source Operation	Operation Based Formatting	Row Based Formatting
Inserts	Before image is null; After image is output.	Only the After image is output.
Updates	Both Before and After images are output.	Only the After image is output.
Deletes	Before image is output; After image is null.	Only the Before image is output.
Truncates	Both Before and After images are null.	Table name is provided, but both Before and After images are null.

Oracle GoldenGate Kafka Replicat also provides granular configuration properties to manage format-specific settings and transformations, so users can tailor the output payload, headers, and metadata to meet specific consumer needs.

For more information on the message formatting definitions, configurations, and sample messages see [Pluggable Formatters](#) documentation.

Implementation and Configuration

The initial setup of the Oracle GoldenGate Kafka Replicat is streamlined. For versions 23.7 and later, the desired output format, for example, JSON and Avro can be selected directly from the format selector drop-down list during the replicat creation process. Further, granular control over message characteristics and behavior is achieved by adding specific configurations within the replicat properties file.

Considerations and Best Practices

Handling PK Updates

A key configuration area involves managing Primary Key (PK) updates. While PK updates are generally discouraged in high-volume replication sources, their occurrence must be carefully managed to ensure accurate data propagation to Kafka.

The Oracle GoldenGate Kafka Replicat leverages the *format.pkUpdateHandling* property to control how these operations are processed. For more information, see [Pluggable Formatters](#)

Table 9-2 Oracle GoldenGate Kafka Replicat PK Update Handling

Setting	Description	Message Output	Behavior
abend	The Replicat process halts and reports an error upon encountering a PK update.	No records produced.	Default

Table 9-2 (Cont.) Oracle GoldenGate Kafka Replicat PK Update Handling

Setting	Description	Message Output	Behavior
delete-insert	The operation is split into two distinct events.	Two Kafka records are produced: one DELETE record (for the old PK) followed by one INSERT record (for the new PK).	Optional. Supported by the <code>json_row</code> formatter for JSON, and typically available for Avro formatters.
update	The PK update operation is treated as a normal update.	A single Kafka record is produced, containing the after image (including the new PK).	Optional. Supported by all formatters.

Example:

```
gg.handler.kafkacconnect.format.pkUpdateHandling=delete-insert
```

The decision to use update or delete-insert must be driven by the design of the downstream consumer applications. Consumers that rely on the message key (the PK) to maintain state in a target data store, for example, a database or data lake) will typically require the delete-insert method to correctly handle the key change and maintain data integrity.

Handling Oracle Number and Date Type Mappings to Avro Types

Oracle GoldenGate Kafka Replicat maps source DATE and NUMBER types as strings by default. While this preserves the data value, it prevents consumers from treating the data as native numeric or timestamp types.

You can leverage logical types to ensure proper mapping:

`enableTimestampLogicalType` (see, [Using the Avro Formatter](#)): Used for mapping source DATE, TIMESTAMP, and TIMESTAMP WITH TIME ZONE types as the Avro `TimestampMicros` logical type (a 64-bit long).

This property must be used together with `gg.format.timestamp`, which defines the exact format mask used by Oracle GoldenGate to read the timestamp data from the trail file.

For example, for a microsecond-level timestamp:

```
gg.handler.kafkahandler.format.enableTimestampLogicalType=true
gg.format.timestamp=YYYY-MM-DD HH:MI:ss.SSS
```

- `enableDecimalLogicalType` (see, [Using the Avro Formatter](#)): Used for mapping source NUMBER types as the Avro `decimal` logical type (a highly-precise bytes primitive).
- The Oracle NUMBER data type is a proprietary numeric data type that supports numbers with up to 38 significant digits of precision. It offers variable precision and scale, precision can range from 1 to 38 and scale up to 164. Precision and scale are required parameters when generating the Avro decimal logical type. This creates a challenge for deterministic schema generation, as Oracle GoldenGate cannot always automatically determine the column's precision and scale, especially when NUMBER is declared without explicit parameters such as, `(NUMBER or NUMBER(*, *))`, which defaults to the maximum precision (38).

- To explicitly enforce precision and prevent data loss, this property is typically used alongside `gg.handler.kafkaconnect.format.oracleNumberScale`, forcing the Avro decimal type to match the expected scale of your source column.

Related Topics

- [Avro Row Formatter](#)
- [The Avro Operation Formatter](#)
- [Operation Data Formatting Details](#)
- [Row Data Formatting Details](#)

9.1.8 Security Considerations and Best Practices

Securing the CDC stream from a source database via Oracle GoldenGate to Kafka requires a comprehensive, multi-layered security strategy. This approach integrates Oracle GoldenGate's internal security features for data at rest and in transit with the robust security protocols inherent to the Kafka ecosystem (such as SSL/TLS and SASL). The security architecture can be segregated into three critical phases: Securing Oracle GoldenGate, Network Transport Security, and Client Authorization.

- [Securing Oracle GoldenGate](#)
- [Network Transport Security \(Oracle GoldenGate to Kafka\) and Client Authorization](#)

9.1.8.1 Securing Oracle GoldenGate

Oracle GoldenGate employs a multi-layered security model to protect sensitive data throughout the real-time replication lifecycle, securing data at rest, in transit, and during administrative access.

- **Data at Rest:** After capture, change records are written to local trail files. For data stored at rest, Oracle GoldenGate supports robust trail file encryption, using strong AES algorithms (AES 128, AES 192, or AES 256), to prevent unauthorized access to persisted change data.
- **Data in Transit:** Protecting data in transit is addressed by securing network communication between Oracle GoldenGate components. The Distribution Path utilizes the Secure WebSocket Protocol (WSS), which can be further enhanced with Mutual TLS (mTLS).
- **Administrative Operations:** All administrative operations—such as configuring and monitoring replication—are securely conducted via REST API calls, protected by TLS 1.3 by default. For authentication, Oracle GoldenGate supports enterprise-grade methods, including client certificates, token-based authentication, and Single Sign-On (SSO) integration with external Identity Providers (IdPs), such as Oracle Identity Cloud Service (IDCS) or Oracle Access Manager (OAM).

For more information on Oracle GoldenGate security details, see Oracle GoldenGate Secure Standards document.

9.1.8.2 Network Transport Security (Oracle GoldenGate to Kafka) and Client Authorization

The security of the replication stream from Oracle GoldenGate to Apache Kafka is governed by the Oracle GoldenGate Kafka Replicat's role as a standard Kafka Producer client. It must

conform to the cluster's security requirements, which are defined by the mandatory security protocol configuration setting.

This protocol dictates the combination of transport encryption and client authentication used for the network connection. Kafka supports four primary protocol modes, summarized in the table below, which range from completely unsecured (PLAINTEXT) to the most robust (SASL_SSL).

Table 9-3 Kafka Transport Security and Client Authentication Protocols

Protocol	Encryption	Authentication
PLAINTEXT	No	No
SSL	Yes (TLS).	Yes (always, via certificate); optional client authentication (via mTLS).
SASL_PLAINTEXT	No.	Yes (SASL)
SASL_SSL	Yes (TLS)	Yes (SASL).

For production environments, Oracle recommends only the TLS-encrypted protocols (SSL and SASL_SSL). The SSL protocol enforces transport encryption (TLS) and can be used for Mutual TLS (mTLS) authentication if the client presents a certificate. Conversely, SASL_SSL is the enterprise standard combining the confidentiality of TLS encryption with the strong, centralized authentication provided by SASL mechanisms, such as PLAIN, Kerberos or SCRAM.

Troubleshooting Connection Issues

Oracle recommends testing the connection with a different producer, for example the native Java producer) from the same OGG instance. During this test, the client should be able to write to the target Kafka cluster successfully. Once the external producer is successful, the same configuration parameters can be readily adapted for the Oracle GoldenGate Kafka Replicat, significantly narrowing the scope of potential configuration errors.

For more information and configuration samples, see [Apache Kafka](#) in Oracle GoldenGate documentation.

9.1.9 Performance Considerations

To improve overall performance, you should review your entire data pipeline, including the Extract processes.

- [Oracle GoldenGate Extract Performance](#)
- [Oracle GoldenGate Kafka Replicat Performance](#)
- [Network Latency](#)

9.1.9.1 Oracle GoldenGate Extract Performance

For best practices and tuning techniques related to Oracle GoldenGate Extract, refer to these resources:

- [Optimizing Network Performance in Hub-Based Oracle GoldenGate Extract Configurations](#)
- [Troubleshooting GoldenGate Extract Performance for Long Batch Transactions](#)
- [Oracle GoldenGate Extract Recovery: Standard vs. Bounded Recovery and How to Avoid Long Restart Times](#)

9.1.9.2 Oracle GoldenGate Kafka Replicat Performance

The performance of the Oracle GoldenGate Kafka Replicat is influenced by configuration settings at two primary points: Producer properties and Oracle GoldenGate Kafka Replicat Configuration.

- [Producer Properties](#)
- [Oracle GoldenGate Kafka Replicat Configuration](#)

9.1.9.2.1 Producer Properties

These are the settings that govern how the Oracle GoldenGate Kafka Replicat sends data to the kafka broker. Several Kafka producer properties can affect performance. The parameters with significant impact are as follows:

- `linger.ms`
- `batch.size`
- `acks`
- `buffer.memory`
- `compression.type`

Considerations and Best Practices

Oracle recommends that you begin by deploying the Oracle GoldenGate Kafka Replicat with the default values for these Kafka producer parameters to establish a performance baseline.

Once the baseline is set, consult the official Kafka documentation to fully understand the role of each parameter. Adjust these settings individually and conduct targeted performance testing to precisely determine the effect of each modification on the end-to-end replication throughput and latency.

9.1.9.2.2 Oracle GoldenGate Kafka Replicat Configuration

The Replicat's configuration settings are critical for optimizing Oracle GoldenGate Kafka Replicat performance.

GROUPTRANSOPS

GROUPTRANSOPS parameter has the single greatest impact on Oracle GoldenGate Kafka Replicat throughput. This parameter controls the number of source operations the replicat buffers and groups together into a larger batch before processing and flushing them. The flush operation, which the Oracle GoldenGate Kafka Replicat calls on the Kafka producer to ensure write durability and subsequently update the checkpoint, is computationally expensive. By increasing the GROUPTRANSOPS setting, the Replicat executes the expensive flush operation less frequently, thus significantly enhancing performance.

Considerations and Best Practices

The default setting for GROUPTRANSOPS is 1,000. Significant performance improvements are often realized by increasing this value, with Oracle recommending up to 20,000 as an optimal starting point for many environments.

Message Modes

The Replicat's message mode, defined by using Transaction Mode (TX) or Operation Mode (OP), also affects performance and data presentation in Kafka. For more details about message modes, see [Oracle GoldenGate Kafka Replicat Message Modes and Maintaining Transactional Boundaries](#).

Considerations and Best Practices

For optimal performance and the most balanced distribution across Kafka partitions, Oracle recommends adopting a table-per-topic pattern, utilizing OP mode, and setting the GROUPTRANSOPS value to 20,000.

The use of TX mode must be carefully evaluated. Since it groups entire source transactions into a single Kafka message, it can frequently lead to larger payloads that may exceed the default Kafka broker limit of 1 MB (message.max.bytes).

9.1.9.3 Network Latency

Another critical factor for stable, high-performance replication is ensuring proper network latency between the Oracle GoldenGate processes. If low latency replication is a priority, Oracle recommends implementing a Mesh deployment topology where Extract process runs close to source database and Oracle GoldenGate Kafka Replicat runs close to target Kafka cluster.

General latency guidelines are presented in the following table:

Table 9-4 Oracle GoldenGate Round-Trip Ping Requirements

Oracle GoldenGate Process	Round-Trip Ping Latency Requirement	Context
Oracle GoldenGate Extract	< 80 ms	Most of the log parsing is done inside the database
MSFT SQL Server Extract	1-2 ms (Ideal)	SQL Server Extract relies on internal CDC inside the SQL Server databases which supplies one row at a time.
Other RDBMS Extracts. For example, PostgreSQL.	40 ms	Standard remote capture
Oracle GoldenGate Kafka Replicat	40ms	Highly dependent on the configured batching frequency and the target system's commit mechanism.

① Note

These latency figures are generic guidelines. You must evaluate and establish appropriate performance baselines based on their specific infrastructure, network topology, and workload characteristics.

9.2 Cloud Object Storage Replication

This chapter describes Cloud Object Storage Replication Best Practices.

- [Introduction](#)
- [Architecture Overview](#)
- [Configuration Considerations and Best Practices](#)
- [DDL Changes and Schema Propagation](#)
- [PK Updates](#)
- [DATE and TIMESTAMP Types](#)
- [Number Types](#)
- [Performance Considerations](#)

9.2.1 Introduction

Oracle GoldenGate Cloud Object Storage Replication is an integral component of the Oracle GoldenGate for Distributed Applications and Analytics (Oracle GoldenGate for DAA) package. It uses a modular handler architecture, enabling flexible and reliable replication of change data from an Oracle GoldenGate trail to target cloud object storage systems including:

- OCI Object Storage
- Azure Data Lake Storage (ADLS)
- Amazon S3 (including S3 API-compatible services)
- Google Cloud Storage
- Microsoft Fabric OneLake

For the latest certified source/target compatibility, see the [Oracle GoldenGate certification matrix](#).

To meet diverse downstream consumption requirements—ranging from data lakes to AI/ML pipelines—the Cloud Object Storage Replicat supports a wide array of industry-standard serialization formats:

- **Parquet:** Optimized for columnar storage and analytical queries.
- **Avro:** Supported in multiple flavors:
 - **Avro Row:** Compact, row-based format
 - **Avro Op:** Verbose format capturing full operation metadata
 - **Avro Object Container File (OCF):** Encapsulated Avro data with embedded schemas
- **Semi-Structured & Flat Files:** Including JSON, XML, and DelimitedText

These format options, combined with configurable file size controls, allow users to align output with their downstream integration and storage standards.

Oracle GoldenGate provides granular control over how data is landed in the object storage to ensure alignment with organizational data standards:

- **Efficient Data Streaming:** Consolidates change data events from multiple source tables into designated buckets or containers through an efficient replication process.
- **Dynamic Partitioning & Mapping:** Automated mapping of database operations to target containers, supporting both row-level and table-level granularity for organized data layouts.

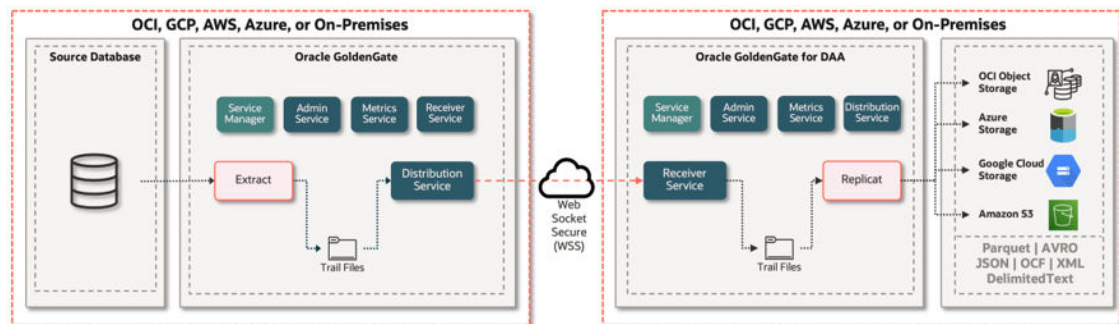
- **Advanced File Lifecycle Management:** Comprehensive features to control file rotation (by size or time), ensuring optimal file sizing for big data processing engines.
- **Rich Data Formatting Options:** An extensive configuration set to define how source database types are translated into target-specific formats, maintaining data integrity across the pipeline.

The following sections provide in-depth coverage of the Cloud Object Storage Replicat architecture, configuration best practices, and guidance for maximizing performance, durability, and data integrity.

9.2.2 Architecture Overview

Oracle GoldenGate Cloud Object Storage Replication architecture is fundamentally a log-based CDC pipeline designed for high-throughput and transactional integrity. As illustrated in the diagram, the architecture bridges relational and non-relational source systems with the Object Storage ecosystem.

Figure 9-2 Oracle GoldenGate Cloud Object Storage Logical Architecture and Key Components



The core design principle is decoupling. The use of the persistent trail files physically separates the Extract process (capture) from the Replicat (delivery). This is a vital architectural advantage because:

- It enables **backpressure tolerance**: If the network to object storage becomes temporarily unavailable or overloaded, the Replicat process can stop and resume seamlessly, while the Extract continues writing new changes to the trail files.
- It **minimizes latency risk**: The high-speed Extract process completes quickly, pushing database responsibility out of the critical path, while the Replicat handles the asynchronous network traffic and serialization overhead independently.

The overall architecture consists of the following major components:

- **Data Sources:** Oracle GoldenGate supports a wide array of transactional systems, including Oracle databases (on-premises and cloud), Microsoft SQL Server, IBM DB2, MySQL, PostgreSQL, Cassandra, MongoDB, Amazon Aurora, JMS, and many more. This flexibility allows enterprises to centralize streaming ingestion from diverse operational systems. For the full list of supported source technologies, you can refer to [Oracle GoldenGate certification matrix](#).
- **Change Data Capture (extract) and Trails:** As change data events are captured, Oracle GoldenGate standardizes all supported source changes into a unified, platform-independent Trail format. These Trails preserve transaction consistency and only emit

committed events, enabling efficient, reliable, and consistent replication to downstream systems. Oracle GoldenGate provides a highly performant and feature-rich technology to detect change data events, especially from the Oracle Database. Oracle GoldenGate's kernel-level integration with Oracle Database allows for the broadest level of feature support and at the highest volumes while at the same time providing the least overhead of Oracle's CDC technologies. For further details, refer to [Oracle GoldenGate certification matrix](#).

- **Oracle GoldenGate Cloud Object Storage Replication:** The Oracle GoldenGate Cloud Object Storage Replication is a purpose-built pipeline to deliver committed change data into target cloud object storage bucket/container with a user-defined file format. The Replicat reads committed change data from Oracle GoldenGate trail files, then maps each transaction to one or more files based on Replicat configuration.

Internally, the Replicat operates in two stages:

1. Replicat creates a local file and starts writing change data to it. When the configured rollover threshold—based on file size, elapsed time, or an inactivity period—is reached, Replicat closes the file.
2. Once the file is closed, it is uploaded to the target object storage. Bucket/container mapping can be dynamically assigned at runtime to support custom partitioning strategies. Files are uploaded only after they are closed (rolled over), so rollover settings directly influence end-to-end data availability latency.

This modular, decoupled architecture enables organizations to build real-time, cloud-native data pipelines with minimal impact on source systems.

9.2.3 Configuration Considerations and Best Practices

The configuration of Oracle GoldenGate Cloud Object Storage Replication involves two core elements:

- **Configuration Properties:** These are key-value settings that govern how data is formatted, mapped, and delivered from GoldenGate to your object storage buckets or containers. These properties manage file format, file roll rules, dynamic filename configurations, and dynamic directory mapping to meet specific operational requirements.
- **Message Formatters:** [Oracle GoldenGate Formatters](#) are modular formatting components that convert change-data operations captured in the GoldenGate trail files into structured messages for object storage targets. Formatters support multiple formats including [JSON](#), [Avro](#) (also used as the schema definition format for Parquet targets), [XML](#) and [DelimitedText](#).

Careful coordination between these two elements is essential to achieve optimal performance, data consistency, and long-term maintainability of the replication pipeline.

- [File Rollover Configuration](#)
- [Partitioning](#)

9.2.3.1 File Rollover Configuration

The Cloud Object Storage Replicat creates a file in the local file system and keeps it open until a rollover condition is met or triggered by a metadata change event from the source (like DDL change).

Users can control file rollover by using four different properties. For more details, see [Configuring the File Writer Handler](#) in Oracle GoldenGate for DAA documentation.

- **Max File Size:** By default, file sizes are configured to 1 GB; but it can be configured as needed. When the max size is reached, file is closed, and a new file is generated.
- **File Roll Interval:** By default, roll interval is not active; but it can be configured as needed. When configured, it starts a timer when file is created. When the interval timing is reached, if the file is still open, it is closed and rolls over to a new one.
- **Inactivity Roll Interval:** By default, inactivity roll is not active; but it can be configured as needed. When configured, it starts a timer to track the inactivity period. Here, inactivity means there are no operations coming from the source system. In other words, there are no CDC data being written to the file. When set, it starts the countdown when the last operation is written to a file. At the end of the countdown, if there are no incoming operations, the file is closed and rolls over to a new one.
- **Roll on Shutdown:** By default, roll on shutdown is set to false; but can be configured as needed. When configured, it closes the open file when the Replicat process is stopped.

Add one or more configurations to Replicat properties file.

Implementation and Configuration

To implement the rollover configuration, set the required property as needed. If more than one rollover condition is configured, the file rolls over when the first condition is met. For example: You set max file size to 1 GB and file roll interval to 1 minute. 1 GB file size is reached in 45 seconds. In this case, Replicat does not wait for 1 minute and rolls over to a new one.

```
gg.handler.<gg.target value>.maxFileSize  
gg.handler.<gg.target value>.fileRollInterval  
gg.handler.<gg.target value>.inactivityRollInterval  
gg.handler.<gg.target value>.rollOnShutdown
```

For example, if you are replicating to OCI Object Storage, at the top of the Replicat properties file you, see `gg.target=oci`.

Sample configurations for OCI Object Storage:

```
gg.handler.oci.maxFileSize=1gb  
gg.handler.oci.fileRollInterval=1m  
gg.handler.oci.inactivityRollInterval=5s  
gg.handler.oci.rollOnShutdown=true
```

For more information on the legal values that can be used with these properties, see *Oracle GoldenGate for DAA documentation*.

Considerations and Best Practices

- Always configure at least two rollover conditions — typically `maxFileSize` combined with either `fileRollInterval` or `inactivityRollInterval` — to handle both high and low volume scenarios.
- **Max File Size:**
 - Avoid very small file sizes, for example a few MBs as they generate excessive small files which degrade performance in big data engines, such as Spark, Hive, and Databricks — known as the **small file problem**.
 - Avoid excessively large files, for example 10GB+ as they increase recovery time on failure and slow down downstream processing.

- Overall, set between **128MB and 1GB** depending on downstream processing engine requirements. For **Parquet/Avro** targets feeding analytics engines, **256MB–512MB** is a common sweet spot.
- **File Roll Interval:**
 - Use when data volumes are low and max file size may never be reached — ensures files are closed and uploaded on a predictable schedule.
 - Avoid very short intervals, for example interval in seconds in high-volume environments as it creates too many small files.
 - For near-real-time pipelines, a roll interval of 5–15 minutes is typically recommended.
 - Combine with `maxFileSize` to ensure files roll on whichever condition is met first.
- **Inactivity Roll Interval:**
 - Always configure this in environments with **intermittent or unpredictable data flows** — without it, a file could remain open indefinitely during quiet periods.
 - Set lower than `fileRollInterval` to ensure quiet periods are handled promptly.
- **Roll on Shutdown:**
 - Always set to true in production — this ensures no data is left stranded in an open local file when the Replicat is stopped for maintenance or restart.

9.2.3.2 Partitioning

The Cloud Object Storage Replicat supports creating partitioned files in target object storage buckets. Partitioning improves downstream pruning and query performance, but over-partitioning increases directory/file counts and amplifies the small-file problem.

To use partitioning, data must be partitioned by the table at the highest level. This is done using the `pathMappingTemplate` property and it should be set to the fully qualified table name. It will create directories within the container/bucket based on the fully qualified source table name. For example:

```
gg.eventhandler.<gg.target value>.pathMappingTemplate=${fullyQualifiedTableName}
```

At runtime, the path resolves to the fully qualified source table name.

For creating partitions, `gg.handler.<gg.target value>.partitioner.<fully_qualified_table_name>` is used. This property can be used with [template keyword](#) such as `${columnValue[column_name]}` or constant string values to define the partition path structure.

Implementation and Configuration

For example, given a source table `SALES.CUSTOMER`, the following configuration partitions files by the `STATE` column in the OCI Object Storage:

```
gg.eventhandler.oci.pathMappingTemplate=${fullyQualifiedTableName}
gg.handler.oci.partitioner.SALES.CUSTOMER=STATE=${columnValue[STATE]}
```


This configuration will create a directory called `SALES.CUSTOMER` in the OCI Object Storage bucket, additional directories within called `STATE=<state_value>` and within that directory, files are created per `STATE`.

```
SALES.CUSTOMER/ STATE=<state_value>/ <files...>
```

Multi-layer partitioning is also possible. For example, given a source table `SALES.CUSTOMER`, the following configuration partitions files by the `STATE` and `CITY` column in the OCI Object Storage:

```
gg.eventhandler.oci.pathMappingTemplate=${fullyQualifiedTableName}  
gg.handler.oci.partitioner.SALES.CUSTOMER=STATE=${columnValue[STATE]}/CITY=${  
columnValue[CITY]}
```

This configuration will create a directory called `SALES.CUSTOMER` in the OCI Object Storage bucket, additional directories within called `STATE=<state_value>`, within each `STATE` directory, sub-directories are created per `CITY` value, and files are written within each `CITY` directory.

```
SALES.CUSTOMER/ STATE=<state_value>/ CITY=<city_value>/ <files...>
```

In a single Replicat, you can use multiple partitioning configurations for several tables.

Considerations and Best Practices

- Always partition by table at the top level before adding column-level partitions — skipping this will cause runtime errors.
- Choose partition columns with **reasonable cardinality** — high cardinality columns like customer ID or timestamp will create too many directories and small files.
- Align partition strategy with downstream query patterns — partition columns should match the most common filter predicates in your queries.
- **Date/time-based partitioning** (e.g. by year/month/day) is the most common and recommended pattern for analytics workloads.
- Avoid partitioning by columns with **null values** unless null handling is explicitly configured.
- In a single Replicat, multiple tables can have independent partitioning strategies — document each table's strategy for maintainability.

9.2.4 DDL Changes and Schema Propagation

The Oracle GoldenGate Cloud Object Storage Replication propagates source DDL changes automatically to the files generated. Alternatively, `EVENTACTIONS` can be used to control the Extract/Replicat behavior in case of a DDL operation in the source database.

Implementation and Configuration

- If Oracle GoldenGate extract captures a `Create Table` event from the source database, the Replicat will create a new file with the table name from the `Create Table` event in the target cloud object storage bucket/container.
- If Oracle GoldenGate extract captures an `Alter Column/Drop Column` event from the source database, the Replicat will create a new file reflecting the updated source table definition.
- If Oracle GoldenGate extract captures a `Truncate` event from the source database, you can configure the Replicat to roll the data file on *Truncate* event by setting

`gg.eventhandler.<gg.target value>.rollOnTruncate=true`. This property is set to false by default and the Replicat does not roll the file in Truncate event.

- If Oracle GoldenGate extract captures a Drop Table event from the source database, this has no impact on the target file system.

If you do not want to propagate the schema changes automatically, you can use `EVENTACTIONS` to control the Extract/Replicat behavior. In this case, `EVENTACTIONS` is used together with `DDL` parameter. For example, if `DDL INCLUDE ALL EVENTACTIONS (LOG INFO, STOP)` is included in the Extract/Replicat parameter file, this includes all the source DDL operations, stop the Extract/Replicat and logs the event.

Considerations and Best Practices

- Always test DDL propagation behavior in a non-production environment before enabling in production - unexpected schema changes can corrupt downstream Parquet/Avro files
- Enable `rollOnTruncate=true` in production — without it, truncated data and new data may coexist in the same file, causing data integrity issues downstream
- Use `EVENTACTIONS` with `DDL` for controlled pipelines — in analytics or data lake pipelines where schema changes require downstream coordination, for example Iceberg schema evolution, Databricks Delta table updates, stopping the Replicat on DDL and handling changes manually is safer than automatic propagation
- Monitor for Alter Column events - column type changes in the source can cause serialization failures in Avro/Parquet targets if schema registry or formatter configuration is not updated accordingly
- Implement object storage lifecycle policies (retention, archival, deletion) and naming conventions that help consumers identify active vs. historical versions.

9.2.5 PK Updates

In the Oracle GoldenGate Cloud Object Storage Replication, primary key update operations require special consideration and planning.

Implementation and Configuration

Based on the target file format, there can be some special considerations; but the Replicat handles PK updates in these modes. This is controlled by adding `gg.handler.name.format.pkUpdateHandling` to the Replicat properties. There are values that can be used with this property, such as `abend`, `update`, and `delete-insert`.

- **Abend:** Default behavior, the Replicat fails when there is a PK update and logs in the report file.
- **Update:** PK update is treated like any other update operation. Use this configuration only if you can guarantee that the primary key is not used as selection criteria.
- **Delete-Insert:** PK update is replicated in two different rows: one with the before image (marked as delete) and one with after image (marked as insert). To use delete-insert, extract process should capture uncompressed change data records meaning that all the columns are written to trail file. To generate uncompressed records in extract process, use `LOGALLSUPCOLS` for Oracle Databases and `COMPRESSUPDATES | NOCOMPRESSUPDATES` for Non Oracle Databases.

For example, while replicating to OCI Object Storage set `gg.handler.<gg.target value>.format.pkUpdateHandling=delete-insert` for replicating PK updates as one delete (before image) and one insert (after image) record.

Considerations and Best Practices

- Use delete-insert when replicating to file-based, analytics, or data lake targets. Formats such as Parquet, Avro, JSON, and other immutable file formats do not support in-place updates. Delete-Insert ensures that both the before and after images are written, allowing downstream processing engines such as Apache Iceberg and Delta Lake to correctly process primary key changes.
- When using delete-insert, configure the Extract process to generate uncompressed update records. Use LOGALLSUPCOLS for Oracle databases and NONCOMPRESSEDUPDATES for non-Oracle databases to ensure that all column values are included in the trail file.
- Evaluate the expected frequency of primary key updates before selecting the handling mode. If primary key updates are not expected, the defaultabend behavior can be used to detect unexpected changes. If primary key updates are expected, delete-insert is the recommended configuration.
- Use update mode only when the target format supports update operations and the primary key is not used as a selection, merge, or partition key in downstream processing. Incorrect use of update mode may lead to inconsistent results.

Related Topics

- [Pluggable Formatters](#)
The pluggable formatters are used to convert operations from the Oracle GoldenGate trail file into formatted messages that you can send to Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) targets using one of the GG for DAA handlers.

9.2.6 DATE and TIMESTAMP Types

Oracle DATE and TIMESTAMP columns are replicated as strings by default when no explicit mapping is configured. Without proper mapping, downstream engines will not recognize these columns as date/time types, making date arithmetic, range filtering, and time-based partitioning impossible without explicit casting. The `enableTimestampLogicalType` property enables mapping of Oracle DATE and TIMESTAMP columns to the Avro `timestamp-micros` logical type, which represents values as microseconds since epoch — the correct representation for analytical and data lake workloads.

Implementation and Configuration

In the Oracle GoldenGate Cloud Object Storage Replication, source DATE and TIMESTAMP types are mapped to Avro files (including Parquet, Iceberg and Delta) as STRING by default. To replicate source DATE and TIMESTAMP types as `timestamp-micros` logical type to Avro files (including Parquet, Iceberg and Delta), set `gg.handler.<gg.target value>.format.enableTimestampLogicalType` to true. When `timestamp` logical type enabled, `gg.format.timestamp` is also required. For example:

```
gg.handler.oci.format.enableTimestampLogicalType=
truegg.format.timestamp=yyyy-MM-dd HH:mm:ss.SSSSS
```

For more information, see *Oracle GoldenGate for Distributed Applications and Analytics* documentation.

Considerations and Best Practices

- Set `enableTimestampLogicalType=true` when replicating to Parquet or Avro targets — without it, timestamps are written as plain strings, losing all temporal semantics and making date-based filtering and partitioning impossible downstream.
- For Oracle source databases, `yyyy-MM-dd HH:mm:ss.SSSSSS` is a common choice when you want microsecond precision.
- `gg.format.timestamp` uses [Java SimpleDateFormat patterns](#)—ensure the pattern matches the timestamp representation produced/expected by the replication process to avoid parse errors at runtime.

9.2.7 Number Types

Oracle `NUMBER` is a variable-precision decimal data type that can represent integers and fixed-point decimal values with up to 38 digits of precision. If a column is defined without explicit precision and scale (for example, `NUMBER`), Oracle can store values with varying scale across rows, and in some replication/schema-generation scenarios the formatter may not be able to deterministically derive a single precision/scale for the target schema. As a result, mappings to Avro/Parquet often use a conservative decimal representation unless you explicitly configure precision and scale to match the source column definitions.

Implementation and Configuration

`gg.handler.<gg.target value>.enableDecimalLogicalType` is the master switch for decimal logical type mapping. When set to `true`, it instructs the Avro formatter to map Oracle `NUMBER` columns to the Avro decimal logical type (bytes with `logicalType: decimal`) instead of the default mapping which writes numbers as plain Avro numeric primitives or strings.

When precision/scale can't be deterministically inferred, the formatter typically chooses a conservative Avro decimal definition (high precision and scale) to avoid precision loss. This can lead to downstream consumers seeing values expressed with an unnecessarily large scale (for example, apparent trailing fractional digits). To align the target schema with known, consistent source definitions—especially for financial data (balances, prices, amounts)—set `gg.handler.<gg.target value>.maxPrecision` and `gg.handler.<gg.target value>.oracleNumberScale`.

For example:

```
gg.handler.oci.enableDecimalLogicalType=true
gg.handler.oci.maxPrecision=38
gg.handler.oci.oracleNumberScale=12
```

For more information on configuration details, see [Pluggable Formatters](#) documentation.

Considerations and Best Practices

- Always set `gg.handler.<gg.target value>.enableDecimalLogicalType=true` for Parquet/Avro-based analytical targets to preserve decimal semantics.
- When source `NUMBER` columns have known, consistent precision and scale — particularly for financial data such as balances, prices, and amounts — set `gg.handler.<gg.target value>.maxPrecision` and `gg.handler.<gg.target value>.oracleNumberScale` to prevent an overly conservative scale in downstream consumers.

- Use `gg.handler.<gg.target value>.mapLargeNumbersAsStrings=true` for streams that mix small integers with very large/high-precision values—this preserves numeric types for typical values while avoiding overflow/compatibility issues for extreme cases.
- Use caution with `gg.handler.<gg.target value>.maxPrecision`: if a source value exceeds the configured maximum precision, Replicat can abend at runtime. Validate source column ranges and test in a non-production environment before rollout.

9.2.8 Performance Considerations

The Oracle GoldenGate Cloud Object Storage Replication performance can be tuned using two key features: Replicat type and GROUPTRANSOPS parameter. In addition to the Replicat type and transaction grouping, the size of the files produced by the Replicat directly affects throughput and should be tuned in conjunction with the other settings.

Oracle GoldenGate Distributed Applications and Analytics provides two different replication modes: Classic Replicat and Coordinated Replicat. Classic Replicat is a single-threaded process that applies the messages to target cloud storage services. Coordinated Replicat is a multi-threaded process where multiple threads read the OGG trail file independently and apply transactions in parallel.

The Oracle GoldenGate Cloud Object Storage Replication process optimizes processing with transaction grouping. The GROUPTRANSOPS parameter groups multiple small transactions into a single larger transaction applied to cloud storage targets. The GROUPTRANSOPS parameter counts the database operations (inserts, updates, and deletes) and only commits the transaction group when the number of operations equals or exceeds the GROUPTRANSOPS configuration setting. GROUPTRANSOPS defers the transaction commit call until the larger transaction is completed. When a transaction is committed, the Replicat flushes the operations.

Implementation and Configuration

Replicat Type is selected at the first step of Replicat creation process. In UI, select the Replicat type as Classic Replicat or Coordinated Replicat.

GROUPTRANSOPS is configured in parameters file. By default, GROUPTRANSOPS is set to 1000. You can increase up to 20000 for better performance. For example:

```
GROUPTRANSOPS 20000
```

For more information on how you can configure the file sizes generated by the Replicat, see [File Rollover Configuration](#) section.

Different performance optimization configurations may have different impact on the server resources. Below chart compares the performance improvements and the impact on the resources. Note that the tests were executed using OCI GoldenGate and 1 OCPU equals to 16GB of memory.

Table 9-5 Performance Improvements and the Impact on the Resources

Replicat Type	GROUPTRANSOPS Settings	Max OCPU*	Performance Improvement
Classic Replicat	1,000	4	1x
Classic Replicat	20,000	4	3x

Table 9-5 (Cont.) Performance Improvements and the Impact on the Resources

Replicat Type	GROUPTRANSOPS Settings	Max OCPU*	Performance Improvement
Coordinated Replicat with 20 threads	1,000	8	5x
Coordinated Replicat with 20 threads	20,000	12	8x

Note

* 1 OCPU is equivalent to 16 GB of memory.

Considerations and Best Practices

- Start with Classic Replicat for initial deployment. Classic Replicat is simpler to configure, monitor, and troubleshoot. Establish a performance baseline with Classic Replicat before moving to Coordinated Replicat, as the added complexity of multi-threading requires careful tuning to realise its benefits.
- Use Coordinated Replicat for high-volume workloads. Based on the performance benchmarks, Coordinated Replicat with 20 threads and GROUPTRANSOPS 20000 delivers up to eight times performance improvement over the baseline Classic Replicat configuration. However, this comes at the cost of higher resource consumption — ensure your environment has sufficient capacity before enabling it.
- Increase GROUPTRANSOPS from the default. The default value of 1,000 is conservative. Increasing to 20,000 delivers up to three times improvement on Classic Replicat with no change in thread count. This is the lowest-cost performance gain available and should be applied in all production deployments.
- Thread count for Coordinated Replicat should match workload characteristics. 20 threads is not a universal recommendation. For workloads with high table count and high parallelism, more threads improve throughput. For workloads with few large tables or strict transaction ordering requirements, fewer threads may be more appropriate. Start with a lower thread count and increase incrementally while monitoring performance.
- Monitor JVM heap alongside performance tuning. Higher GROUPTRANSOPS values and more Coordinated Replicat threads both increase JVM memory consumption. Ensure jvm.bootoptions heap settings (-Xmx and -Xms) are sized appropriately to avoid OutOfMemory errors under peak load.
- Change one knob at a time and measure. Tune one variable per test cycle—for example GROUPTRANSOPS, Coordinated Replicat thread count, or file rollover sizing/intervals—and measure throughput, end-to-end latency, CPU, and JVM heap impact before making the next change. This avoids multi-variable tuning confusion and makes it easier to attribute performance gains or regressions. Validate changes in a representative non-production environment before applying to production.
- Test performance tuning changes in non-production first. Performance configurations interact with each other and with source transaction patterns. Always validate changes in a representative non-production environment before applying to production pipelines.

9.3 Snowflake

The Snowflake Replicat delivers changed data from Oracle GoldenGate trail files into Snowflake target tables.

This chapter describes Oracle GoldenGate for Snowflake Replication Best Practices.

- [Introduction](#)
- [Architecture Overview](#)
- [Configuration Considerations and Best Practices](#)
- [Snowflake Schema Change Handling](#)
- [LOB Type Replication](#)
- [Snowflake Apache Iceberg Tables](#)
- [Handling PK Updates for Snowflake Stage and Merge Replicat](#)
- [Oracle GoldenGate Snowflake Performance Considerations](#)

9.3.1 Introduction

Oracle GoldenGate for Distributed Applications and Analytics (GoldenGate for DAA) supports replication into Snowflake, enabling changed data from GoldenGate trail files to be delivered into Snowflake target tables. Snowflake can run on AWS, Microsoft Azure, or Google Cloud Platform, and GoldenGate provides two replication options depending on the workload and latency requirements.

The first option is the [Snowflake Stage and Merge Handler](#). This is the general-purpose approach for Snowflake replication and supports workloads that include inserts, updates, deletes, and other change operations. GoldenGate formats trail records into staged files, places them in a staging location, and then applies the changes to Snowflake target tables using SQL operations such as COPY INTO and MERGE. The staging location can be a Snowflake internal stage or an external object store such as Amazon S3, Azure Data Lake Storage Gen2, or Google Cloud Storage.

The second option is the [Snowflake Streaming Handler](#). This handler uses the Snowpipe Streaming API to load rows directly into Snowflake with lower latency and without an intermediate staging area. It is designed for insert-only workloads. If the replication use case includes updates or deletes, the Stage and Merge Handler is usually the better fit unless those operations are intentionally converted into insert records.

At a high level, use Stage and Merge when you need full change data capture behavior and broader workload support. Use Streaming when the workload is insert-only and the priority is lower-latency delivery into Snowflake.

Table 9-6 Differences between Stage and Merge, and Streaming Handlers

Details	Stage and Merge	Streaming Handler
Workload	Any (insert, update, delete, truncate)	Inserts-only
Latency	Micro-batch	Real-time
Snowflake Compute	Customer-managed virtual warehouse	Snowflake-managed serverless compute

Table 9-6 (Cont.) Differences between Stage and Merge, and Streaming Handlers

Details	Stage and Merge	Streaming Handler
API	JDBC+SQL commands (PUT/COPY INTO/MERGE)	Snowpipe Streaming using snowflake-ingest-sdk
Load method	Requires intermediary loading step (internal or external stage).	Direct loading into Snowflake

9.3.2 Architecture Overview

Oracle GoldenGate supports exactly-once semantics for both Snowflake Stage and Merge and Snowflake Streaming replication. For Stage and Merge, Oracle GoldenGate uses checkpointed micro-batch processing and merge application to recover without duplicating committed work. For Snowflake Streaming, Oracle GoldenGate uses Snowpipe Streaming for supported insert-only workloads, where offset-based recovery preserves exactly-once ingestion semantics.

Snowflake Stage and Merge Replication:

The Snowflake Stage and Merge Handler is the primary GoldenGate for DAA architecture for replicating full change data capture workloads into Snowflake. It supports insert, update, delete, and truncate operations — including key changes — by reading Oracle GoldenGate trail records, staging changed data in micro-batches, and then applying those changes to Snowflake target tables through merge processing.

Before data is staged, Oracle GoldenGate performs operation aggregation. Multiple operations on the same source record can be compressed into the final effective operation to reduce unnecessary work and improve apply performance. For example, an insert followed by several updates on the same row can be collapsed into a single insert containing the final row image. The operation aggregation window is 3 minutes by default and can be configured based on latency, memory, and throughput requirements.

The staged data is written as Avro files to either a Snowflake internal stage or an external cloud object store such as Amazon S3, Azure Data Lake Storage Gen2, or Google Cloud Storage. Staged files are rolled — that is, closed and made available for processing — on a configurable micro-batch interval. For external stages, Oracle GoldenGate creates external tables over the staged Avro files so Snowflake can read the staged records before applying them to the target tables.

The high-level flow is:

1. Oracle GoldenGate Replicat reads changed records from trail files.
2. Operation aggregation combines multiple changes on the same record where possible.
3. Changed records are staged as Avro files in an internal or external stage.
4. For external stages, external tables are created over the staged files.
5. Snowflake merge operations apply the staged changes to the target tables.

This architecture is best suited for general-purpose Snowflake replication where source changes must preserve normal CDC semantics, including updates and deletes.

Snowflake Streaming

The Snowflake Streaming Handler provides a lower-latency architecture for insert-only replication into Snowflake. Instead of writing staged files and later merging them into target tables, Oracle GoldenGate reads records from the trail and streams them directly to Snowflake using the Snowpipe Streaming API.

Because this flow loads rows directly into Snowflake, it removes the intermediate staging step used by Stage and Merge. This makes it a better fit for append-only workloads where near real-time delivery is more important than full update/delete merge semantics.

For supported insert-only workloads, Oracle GoldenGate Snowflake Streaming supports exactly-once semantics by using Snowpipe Streaming and offset-based recovery behavior during retry and recovery scenarios.

The high-level flow is:

1. Oracle GoldenGate Replicat reads records from trail files.
2. The Snowflake Streaming Handler sends rows through the Snowpipe Streaming API.
3. Snowflake ingests the records directly into the target table.

This architecture is best suited for insert-only or append-only workloads where near real-time latency is the priority.

9.3.3 Configuration Considerations and Best Practices

The configuration of Oracle GoldenGate Snowflake replication involves two core elements:

- **Configuration Properties:** Key-value settings that govern how data is formatted, mapped, staged, and delivered from GoldenGate to Snowflake tables. These control the Snowflake JDBC connection, the number of concurrent JDBC connections to the target Snowflake database, the operation aggregation window, connection retry behavior, merge behavior (SQL MERGE or delete-and-insert processing), and the Snowflake table type (native Snowflake tables or Iceberg tables).
- **Parameters Configuration:** Defines source-to-target table mappings and applies Oracle GoldenGate parameters for data reformatting use cases such as column filtering, token substitution, and SQL conversion.

Careful coordination between these two elements is essential. For example, a mismatch between a column mapping defined in the parameters file and the actual target table schema can cause apply errors at merge time that are difficult to diagnose without clear logging.

- [Snowflake Session and Connection Stability](#)
Oracle GoldenGate uses Snowflake JDBC for connections. This connection might be interrupted for several reasons, and it can be controlled Snowflake JDBC options and replicat properties.
- [Snowflake Authentication](#)
Oracle GoldenGate supports both username-password and key-pair authentication for Snowflake Stage and Merge replication. For Snowflake Streaming replication, key-pair authentication is required.
- [Precise Instantiation for Snowflake](#)

9.3.3.1 Snowflake Session and Connection Stability

Oracle GoldenGate uses Snowflake JDBC for connections. This connection might be interrupted for several reasons, and it can be controlled Snowflake JDBC options and replicat properties.

Implementation and Configuration

On the Snowflake JDBC level, it is recommended to use [CLIENT_SESSION_KEEP_ALIVE](#) parameter to keep the session active even if there is no activity. In the Snowflake JDBC string, it should be set to true. For example:

```
jdbc:snowflake://<account>.snowflakecomputing.com/?  
warehouse=<warehouse>&db=<database>&schema=<schema>&CLIENT_SESSION_KEEP_ALIVE=  
true
```

For Snowflake Stage and Merge replication, optionally you can set configuration properties for retries.

`gg.eventhandler.snowflake.connectionRetries` property allows you to set the number of times connections to the target data warehouse will be retried.
`gg.eventhandler.snowflake.connectionRetryIntervalSeconds`. `connectionRetries` sets the delay in seconds between connection retry attempts.

For example:

```
gg.eventhandler.snowflake.connectionRetries=5  
gg.eventhandler.snowflake.connectionRetryIntervalSeconds=30
```

Considerations and Best Practices

- Set `CLIENT_SESSION_KEEP_ALIVE=true` in the Snowflake JDBC connection string. GoldenGate Replicat processes can run for long periods with limited Snowflake activity, especially during low-volume periods or while the operation aggregation buffer is filling. This setting helps keep the Snowflake JDBC session active and reduces the risk of session timeout before the next apply operation.
- **Configure connection retries for production deployments:** Transient Snowflake connectivity issues, such as warehouse scaling events, brief network interruptions, or cloud platform maintenance, can cause temporary connection failures that do not require operator intervention. A starting point of 5 retries with a 30-second interval is appropriate for most environments, but these values should be adjusted based on observed Snowflake warehouse restart times and network stability.
- **Monitor GoldenGate Replicat reports, logs, and operational alerts for Snowflake connection errors and retry attempts:** Repeated retries may indicate a persistent network, warehouse, authentication, or session-management issue. Configure alerting for Replicat abends so connectivity problems are detected quickly and do not silently increase replication lag.

9.3.3.2 Snowflake Authentication

Oracle GoldenGate supports both username-password and key-pair authentication for Snowflake Stage and Merge replication. For Snowflake Streaming replication, key-pair authentication is required.

Oracle GoldenGate supports both username-password and key-pair authentication for Snowflake Stage and Merge replication. For Snowflake Streaming replication, key-pair authentication is required.

Implementation and Configuration

Username-Password Authentication

Username-password authentication is supported for Snowflake Stage and Merge replication only. It is suitable for development and testing, but key-pair authentication is recommended for production deployments.

The Snowflake username and password are configured using the following Replicat properties:

```
gg.eventhandler.snowflake.UserName=<snowflake_username>
gg.eventhandler.snowflake.Password=<snowflake_password>
```

Never store credentials in plaintext in the Replicat properties file. Instead, store them securely in the [Using Identities in Oracle GoldenGate Credential Store](#). You can create the Credential Store entry from the Oracle GoldenGate UI by navigating to **Administration Service , DB Connections** and then **Add DB Connection** or by using the [Oracle GoldenGate Rest API](#).

When creating the entry, provide a Credential Domain and Credential Alias, enter the Snowflake username as the User ID, and the Snowflake password as the Password. Oracle GoldenGate stores these values securely.

Once stored, reference the credentials in the Replicat properties file using the domain and alias you defined:

```
gg.eventhandler.snowflake.UserName=ORACLEWALLETUSERNAME[alias domain_name]
gg.eventhandler.snowflake.Password=ORACLEWALLETPASSWORD[alias domain_name]
```

Where *<alias>* is the Credential Alias and *<domain>* is the Credential Domain you assigned when creating the entry.

Key-Pair Authentication

Key-pair authentication is supported for both Snowflake Stage and Merge and Snowflake Streaming replication and is the recommended authentication method for production deployments.

Oracle GoldenGate uses Snowflake-recommended PKCS#8 .p8 private keys. For supported cryptographic key types and algorithms, refer to the [Snowflake key-pair authentication documentation](#).

For Snowflake Stage and Merge replication, key-pair authentication is configured in the Snowflake JDBC connection string. The Snowflake user and private-key file password can be referenced securely from the [Using Identities in Oracle GoldenGate Credential Store](#). You can create the Credential Store entry from the Oracle GoldenGate UI by navigating to **Administration Service , DB Connections** and then **Add DB Connection** or by using the [Oracle GoldenGate Rest API](#).

When creating the entry, provide a Credential Domain and Credential Alias. Enter the user as the User ID, and the private_key file password as the Password. Oracle GoldenGate stores these values securely.

Once stored, reference the credentials in the In Snowflake JDBC string:

```
jdbc:snowflake://<account_name>.snowflakecomputing.com/?warehouse=<warehouse-
name>&db=<database-name>&role=<role>&
user=ORACLEWALLETUSERNAME[alias domain]
&private_key_file=/path/to/key/file/rsa_key.p8&
private_key_file_pwd=ORACLEWALLETPASSWORD[alias domain]
```

For more information on the configuration steps, see, [Snowflake Key Pair Authentication](#) and [Snowflake Streaming Handler Key Pair Authentication](#).

Considerations and Best Practices

- **Use key-pair authentication for production deployments:** Username-password authentication should be reserved for development and testing environments. Key-pair authentication eliminates password expiry as a source of replication outages and better aligns with enterprise security policies.
- **Restrict private-key file permissions:** The private-key file should be readable only by the OS user running the GoldenGate Replicat process.
- **Never store plaintext credentials in the Replicat properties file:** Use the Oracle GoldenGate Credential Store for username-password authentication. For key-pair authentication, ensure the referenced private-key file has restricted OS-level permissions.
- **Verify JDBC driver compatibility:** Many key-pair authentication issues stem from JDBC driver version mismatches or unsupported key algorithms. Confirm that the Snowflake JDBC driver bundled with your GoldenGate installation supports the key type and algorithm before deploying to production.
- **Plan for zero-downtime ,[Snowflake key rotation](#):** Snowflake allows a secondary public key to be assigned to a user, enabling key rotation without interrupting replication. Register the new public key as the secondary key in Snowflake, update Oracle GoldenGate to reference the new private key, confirm replication, and then remove the old public key from Snowflake.

9.3.3.3 Precise Instantiation for Snowflake

Oracle GoldenGate supports precise instantiation for databases that support Oracle GoldenGate Initial Load Extract. For Snowflake targets, precise instantiation enables a seamless transition from initial load to CDC replication without downtime or data loss. The configuration depends on the source database type. This section covers Oracle Database as the source; for other supported source databases, contact an Oracle engineer or Oracle Support.

Implementation and Configuration

Oracle GoldenGate Snowflake Replicat supports automated table creation in Snowflake. During startup, the Replicat checks whether the target table exists. If the target table does not exist, Oracle GoldenGate automatically creates it using the source table definition and the *default Oracle-to-Snowflake data type mapping*. If the target Snowflake table already exists, the Replicat uses the existing target table definition and data mapping.

Precise instantiation from Oracle Database to Snowflake can be configured with the following steps using the Oracle GoldenGate UI with the following steps:

- Create and run the primary CDC extract.
- Create a Distribution Path for the CDC extract.
- Determine the Instantiation SCN using SQL*Plus

Create and run the Initial Load Extract using SQLPREDICATE "AS OF SCN" in the parameter file. For example:

```
EXTRACT <process_name> EXTFILE <trail_name> USERIDALIAS <alias> DOMAIN
<domain>
TABLE <schema>.*; SQLPREDICATE "AS OF SCN <scn_value>
```

- Create a Distribution Path for the Initial Load Extract.
- Create and run the Initial Load Replicat for Snowflake.
 - Initial Load Replicat should use the trail file generated by the Initial Load Extract.
 - Use [INSERTALLRECORDS Support](#) to improve initial load performance.
- Create the CDC Replicat for Snowflake using the trail file generated by CDC Extract. Create the CDC Replicat, but do not start it yet
- Start the CDC replicat using Start with Options:
 - **Start Position:** After SCN
 - **CSN:** The SCN value used in Initial Load Extract

Considerations and Best Practices

- **Run the CDC Extract before the Initial Load Extract:** The CDC Extract must be capturing changes before the Initial Load begins. This ensures no changes made during the initial load window are missed. Starting the CDC Extract after the initial load risks a gap between the instantiation SCN and the point where CDC begins, leading to data loss or requiring a restart.
- **Validate the instantiation SCN before proceeding:** Before creating the Initial Load Extract, confirm that the SCN retrieved from SQL*Plus is usable for flashback query and that sufficient undo retention is available for the expected duration of the initial load. The Initial Load Extract uses this SCN to read a consistent point-in-time image of the source data, and the CDC Replicat must later be started from the same value to avoid gaps or duplicates. For more information, see Refer to Oracle GoldenGate documentation for details.
- **Do not start the CDC Replicat until the Initial Load Replicat has finished:** Starting the CDC Replicat before the Initial Load Replicat completes can apply change records before the corresponding base rows are loaded, which may cause missing-row conditions or incorrect target state. Monitor the Initial Load Replicat until it reaches end of trail before starting CDC.
- **Use automated table creation with caution:** The auto-create feature is convenient but relies on Oracle-to-Snowflake default type mappings, which may not match your target schema requirements, for example, precision on numeric types, VARCHAR sizing, or timestamp handling. If your Snowflake tables have been pre-created with custom definitions, ensure the Replicat is pointed at an existing table and validate the column mapping before starting replication.
- **Document and retain the SCN value used for each instantiation:** The SCN is the key reference point linking the initial load to CDC. Store it alongside the instantiation date, source table scope, and Oracle GoldenGate process names involved. This record is essential for troubleshooting, auditing, or repeating the instantiation for additional tables at a later stage.

9.3.4 Snowflake Schema Change Handling

Oracle GoldenGate Snowflake Streaming Replicat and Snowflake Stage Merge Replicat support propagation of source schema changes to target Snowflake tables with Automatic Schema Evolution functionality.

Automatic schema evolution is the capability to detect, manage, and apply changes to a database schema — such as tables, columns, indexes, or relationships — without requiring manual intervention.

Implementation and Configuration

Automatic Schema Evolution for Snowflake automatically creates and evolves Snowflake target tables based on the source database schema when *AUTOSCHEMA* is enabled.

When *AUTOSCHEMA* is enabled, Oracle GoldenGate automatically creates the target Snowflake using the source table definition and the *default Oracle-to-Snowflake data type mapping*. If your Snowflake tables have been pre-created with custom definitions, Oracle GoldenGate Snowflake Replicat will use the target Snowflake table definition for mapping.

As an alternative to automatic schema evolution, source schema changes can be monitored by enabling DDL capture in Extract and using DDL with *EVENTACTIONS* to report the DDL event, stop the process, or trigger an operational response. For example, `DDL INCLUDE ALL EVENTACTIONS (REPORT, STOP)` captures source DDL, writes it to the report file, and stops the process so the target schema can be reviewed before replication continues.

Considerations and Best Practices

- **Enable Automatic Schema Evolution carefully because schema changes can affect downstream applications:** Source changes such as adding, renaming, or dropping columns may be propagated to Snowflake without manual review. Downstream consumers such as BI tools, dbt models, and data pipelines may fail if a column they depend on is changed or removed.
- **Use DDL with `EVENTACTIONS` as a controlled alternative to AUTOSCHEMA for production environments.**
- **Align schema change procedures with Snowflake governance policies:** In environments where schemas are managed through a data catalog, schema registry, or infrastructure-as-code process, automatically propagated schema changes may create drift from the governed definition.

9.3.5 LOB Type Replication

Oracle GoldenGate supports replication of Large Object (LOB) columns, including BLOB, CLOB, and NCLOB, from Oracle Database to Snowflake targets.

Implementation and Configuration

For LOB column replication to Snowflake, Oracle GoldenGate Integrated Extract must be configured with `TRANLOGOPTIONS FETCHPARTIALLOB` in the Extract parameter file. Oracle GoldenGate Snowflake Replicat does not support partial LOB data. Without this option, transactions containing LOB columns that were not fully captured in the redo log can cause the Replicat toabend.

LOB columns that exceed the inline LOB threshold are processed out-of-band, which can lead to Replicat performance degradation. The `gg.maxInlineLobSize` property controls the maximum size, in bytes, of LOB data processed inline. The default value is 16000 bytes. Increasing this value allows larger LOB column updates to be processed within the standard micro-batch flow rather than individually out-of-band.

Before setting this value, check the trail files containing LOB data or review the source table definitions to determine the actual maximum LOB size in your dataset. For example, to process LOBs up to 24 MB inline: `gg.maxInlineLobSize=24000000`

For details about default Oracle LOB type mappings to Snowflake types, refer to *Oracle-to-Snowflake data type mapping documentation*.

Considerations and Best Practices

- **Always configure Oracle Extract with *TRANLOGOPTIONS FETCHPARTIALLOB* when replicating LOB columns.**
- **Measure actual LOB sizes before configuring the Snowflake replicat with LOB type and tune it using *gg.maxInlineLobSize*:** The inline LOB buffer is allocated per Replicat process, and setting it significantly higher than the actual maximum LOB size in your dataset wastes memory. Query the source table or inspect trail file statistics to determine the realistic maximum LOB size, then set *gg.maxInlineLobSize* to match or slightly exceed that value.
- **Validate target Snowflake column types for LOB data before go-live:** BLOB data maps to BINARY(67108864) and CLOB/NCLOB map to STRING(134217728) by default. Verify that downstream consumers of the Snowflake table can handle these types correctly.
- **Exclude LOB columns from replication if they are not required at the target.** LOB replication adds overhead to both the Extract (due to out-of-row fetching) and the Replicat (due to increased trail file size and apply time). If downstream consumers do not require LOB data, use *COLSEXCEPT* in the Extract parameter file to exclude LOB columns from replication.
- **Consider whether LOB-heavy tables warrant a dedicated Extract and Replicat process to isolate their impact on the broader replication pipeline.**

9.3.6 Snowflake Apache Iceberg Tables

Oracle GoldenGate Snowflake Streaming Replicat and Snowflake Stage Merge Replicat support Snowflake managed Apache Iceberg table type. [Snowflake Open Catalog](#) is supported Oracle GoldenGate Apache Iceberg Rest Catalog Replicat.

Implementation and Configuration

Snowflake Managed Iceberg Tables

For Snowflake-managed Apache Iceberg tables, the `tableType` property must be set to `iceberg`.

For example, in the Snowflake Stage and Merge Replicat:

```
gg.eventhandler.snowflake.tableType=iceberg
```

When `tableType` is set to `iceberg`, the following additional properties are required:

- `icebergExternalVolume`: The name of the external volume that connects Snowflake to your external cloud storage for Iceberg tables. The external volume must be [configured in Snowflake](#) before any Iceberg tables can be created.
- `icebergBaseLocation`: The base path in external storage where Iceberg table data and metadata will be written. Oracle GoldenGate appends the qualified table name to this path when creating each Iceberg table.

Example:

```
gg.eventhandler.snowflake.tableType=iceberg
gg.eventhandler.snowflake.icebergExternalVolume=<external_volume_name>
gg.eventhandler.snowflake.icebergBaseLocation=<base_location_path>
```

Snowflake Open Catalog

For replicating Apache Iceberg tables managed by Snowflake Open Catalog, use [Compressed Update Handling](#). Snowflake Open Catalog does not require storage authentication at the Replicat level.

Configure the following properties in the Replicat properties file:

```
gg.eventhandler.iceberg.restCatalogUri=<open_catalog_rest_uri>
```

```
gg.eventhandler.iceberg.restCatalogProperties=/path/to/opencat.properties  
gg.classpath=/path/to/required/dependencies/*
```

Create a catalog.properties file (for example, opencat.properties) to be referenced in the Replicat properties file:

```
type=rest warehouse=<warehouse_name>  
credential=secret:key scope=PRINCIPAL_ROLE:ALL
```

For enhanced security, secure the catalog properties file using the encryption capabilities supported by *Oracle GoldenGate Apache Iceberg Rest Catalog Replicat*.

Considerations and Best Practices

- **Configure and validate the external volume before starting the Replicat:** If the external volume is missing or misconfigured, the Replicat will fail to create Iceberg tables during startup. Validate the external volume in Snowflake using `SHOW EXTERNAL VOLUMES` and confirm that Snowflake has the necessary permissions to read and write to the underlying cloud storage location before running the Replicat for the first time.
- **Set icebergBaseLocation carefully — it cannot be changed after tables are created.** Oracle GoldenGate appends the qualified table name to the base location path to construct the storage path for each Iceberg table. Once an Iceberg table has been created at a given path, changing the base location requires dropping and recreating the table and performing a full re-instantiation of the affected tables. Choose a base location path that is consistent with your cloud storage naming conventions and organisation structure before go-live.
- **For Open Catalog deployments, secure the catalog.properties file:** Restrict file permissions so the file is readable only by the operating system user running the Replicat process. If the file contains catalog credentials, use Oracle GoldenGate-supported encryption or secret-management controls.
- **Monitor Iceberg metadata and snapshot growth:** Iceberg tables generate metadata and snapshot files over time. Establish a maintenance strategy with the Snowflake administration team to manage metadata growth, snapshot retention, and storage costs for Oracle GoldenGate-replicated Iceberg tables

9.3.7 Handling PK Updates for Snowflake Stage and Merge Replicat

In Snowflake Stage and Merge Replicat, primary key updates require special attention because the key value used to identify the target row changes during the operation.

For primary key update operations, if the trail record is compressed or does not contain all required column values, Oracle GoldenGate may need to apply the PK update as a singleton operation — a row-by-row update applied outside the MERGE batch. This bypasses operation aggregation and significantly impacts performance in PK update-heavy workloads.

Implementation and Configuration

For uncompressed update records, configure the source extract using `LOGALLSUPCOLS` for Oracle Database extracts and `COMPRESSUPDATES | NOCOMPRESSUPDATES` for Non-Oracle Database extracts.

In the Snowflake Stage and Merge replicat properties, set the following properties:

```
gg.compressed.update=false  
gg.validate.keyupdate=true
```

These settings will allow replicat to aggregate PK update operations and improve the performance, especially for PK update heavy workloads.

For workloads with frequent primary key updates, consider enabling SQL-based operation aggregation when the trail contains uncompressed update records. SQL Aggregation allows Oracle GoldenGate to aggregate operations using SQL before the Snowflake MERGE, which can improve throughput and reduce Replicat JVM memory pressure. To enable SQL based operation aggregation, set the following property:

```
gg.aggregate.operations.using.sql=true
```

Considerations and Best Practices

- **Ensure the trail format has uncompressed operations before starting the Replicat:** If `gg.compressed.update=false` is set but the trail contains compressed updates, the Replicat will encounter column mapping mismatches at apply time. Validate the trail format using logdump before changing this property in a running deployment.
- **Enable `gg.validate.keyupdate=true` whenever PK updates are present in the source workload:** Some source applications generate PK update operations where the key value does not actually change — for example, as a result of application-level row refresh logic. Without this property, these false PK updates are treated as genuine key changes and cannot be aggregated into the MERGE batch, degrading performance unnecessarily. This property corrects the operation type before aggregation, improving both accuracy and throughput.
- **Assess the source workload for PK update frequency before tuning:** PK updates are relatively rare in most OLTP workloads. If PK updates represent a small fraction of total DML, the performance impact of singleton fallback may be negligible and the overhead of changing the trail format (re-running Extract with `LOGALLSUPCOLS` or `NOCOMPRESSUPDATES`) may not be justified. Measure the proportion of PK update operations in the trail using logdump or GoldenGate statistics before committing to uncompressed trail configuration.

9.3.8 Oracle GoldenGate Snowflake Performance Considerations

Oracle GoldenGate Snowflake replication performance depends on the selected delivery mode, Snowflake warehouse capacity, target table design, micro-batch sizing, connection parallelism, workload characteristics, and Oracle GoldenGate/Snowflake JDBC driver versions. Most performance tuning requirements are related to frequent small batches, stage-file backlog, throttling, LOB processing, insufficient JDBC connection concurrency, or metadata-query overhead in older software versions.

Implementation and Configuration

Snowflake Stage and Merge Micro-batch window control

Oracle GoldenGate Snowflake Stage and Merge Replicat is a micro-batch process. Change data from Oracle GoldenGate trail files is staged in micro-batches at a temporary staging location, either an internal Snowflake stage or an external cloud object store such as S3, ADLS Gen2, or GCS. The staged data is then merged into Snowflake target tables using SQL *MERGE* statements.

The micro-batch window is controlled by the `gg.handler.snowflake.fileRollInterval` property. The default value is 3 minutes.

For high-volume replication, increasing the micro-batch window is recommended. Small staged files result in many small Snowflake *MERGE* statements, which can increase both latency and cost because each *MERGE* may scan more data than the number of changed rows. A larger `fileRollInterval` allows Stage and Merge Replicat to aggregate more operations per batch, resulting in fewer and more efficient *MERGE* operations.

For example, to set a 15-minute micro-batch window:

```
gg.handler.snowflake.fileRollInterval=15m
```

As part of the micro-batch process, GoldenGate Snowflake Stage and Merge Replicat runs operation aggregation. Operation aggregation reduces the number of rows written to the stage file and merged into Snowflake by collapsing multiple operations on the same row within a micro-batch into a single net operation. For example, multiple updates to the same row within the batch window are collapsed into one operation, reducing the number of rows in the staged file and the number of rows processed by the *MERGE* statement.

Operation aggregation requires additional JVM heap allocation. Configure the JVM heap size in the replicat properties file to accommodate the expected volume of in-flight operations within the micro-batch window.

For example:

```
jvm.bootoptions=-Xmx8g -Xms8g
```

The appropriate heap size depends on the number of tables, the volume of changes per micro-batch, and the average row size. Monitor JVM garbage collection behavior after tuning, because GC pauses during micro-batch aggregation can appear as apply-lag spikes.

Considerations and Best Practices

- **Choose the appropriate Snowflake delivery mode that works best for your requirement** : Use Snowflake Stage and Merge replication for mix or update intense workloads. For low-latency and insert only requirements, evaluate Snowflake Streaming replication when the workload and supported operations fit the use case.
- **Tune `fileRollInterval` based on workload volume and latency requirements to avoid frequent small *MERGE* operations.** : The default 3-minute interval is a starting point, not a production target. For high-volume workloads, a longer interval — 10 to 30 minutes — produces larger, fewer stage files and more efficient *MERGE* operations. For low-latency requirements, a shorter interval reduces end-to-end replication lag but increases the number of *MERGE* statements and Snowflake warehouse compute consumption. Balance the two based on your SLA and Snowflake cost targets. Avoid values below 30 seconds in production — very short flush intervals generate excessive small Snowflake transactions, which can paradoxically increase latency under load.

- **Parallelize large initial loads by splitting the workload across multiple Initial Load Extract and Replicat processes.** This is particularly effective for large schemas or tables with high row counts. For configuration details, refer to the Precise Instantiation section.
- **Tune LOB handling intentionally:** If your replicat includes LOB types, tune Oracle GoldenGate Snowflake Replicat to handle LOB type replication more efficiently. For more details, see [LOB Type Replication](#).
- **Measure end-to-end latency.** It is possible to measure the end-to-end latency between when a transaction was committed on the source and when it was committed on the target is a common requirement using extra columns on the target to store the operation type performed by GoldenGate, the source commit timestamp from the trail, and the target commit timestamp using the CURRENT_TIMESTAMP() function as its default value. For configuration details and sample codes, refer to [measuring-the-endtoend-lag-using-the-snowflake-handler](#).
- **Handle PK updates carefully:** For PK update-heavy workloads, always use uncompressed update records and SQL-based operation aggregation to enable efficient MERGE processing and avoid singleton row-by-row apply. For configuration information, see [Handling PK Updates for Snowflake Stage and Merge Replicat](#).
- **Co-locate the Oracle GoldenGate Snowflake Replicat process and target Snowflake instance in the same cloud provider and in the same region:** Co-locating Oracle GoldenGate Snowflake Replicat and the Snowflake instance in the same cloud provider and in the same region will optimize network latency.
- **Upgrade to GoldenGate for DAA 26ai to take advantage of optimized aggregation:** The redesigned aggregation engine in 26ai delivers better throughput with lower resource consumption and enables larger batch windows without proportionally increasing JVM memory requirements.

10

Replicate Data

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) supports specific configurations - the handlers (which are compatible with clearly defined software versions) for replicating data.

Handlers in GG for DAA are components that manage the data flow between various sources and targets. They are responsible for reading data from sources such as databases, log files, or message queues, and writing the data to a wide range of target systems. GG for DAA uses handlers to perform various tasks, such as data ingestion, data transformation, and data integration. Handlers are essential for enabling real-time data movement and data replication across technologies.

Note

GG for DAA includes utility scripts called dependency downloader. The dependency downloader helps you to download required client libraries. The dependency downloader is located at `GG_HOME/opt/DependencyDownloader`.

To be able to use dependency downloader, please install JDK in your instance and set the following environment variables:

```
echo $JAVA_HOME should point to GG_Home/jdk
echo $LD_LIBRARY_PATH should point GG_HOME/jdk/lib/server
echo $PATH should include GG_HOME/jdk/bin
```

For more information, see [Dependency Downloader](#).

This article describes the following Sources and Target Handlers in GG for DAA:

- [Source](#)
- [Target](#)

10.1 Source

The Extract process is configured to run against the source technology, capturing data generated in the true source technology located somewhere else. This process is the extraction or the data capture mechanism of GG for DAA.

You can configure an Extract for the following use cases:

- **Initial Load Extract:** When you set up GG for DAA for initial loads, the Extract process captures the current, static set of data directly from the source objects. This configuration of Extract process uses source source to capture data.
- **Change Data Capture Extract:** When you set up GG for DAA to keep the source data synchronized with another set of data, the Extract process captures the DML and (if supported) DDL operations performed on the configured objects after the initial synchronization has taken place. It stores these operations until it receives commit records or rollbacks for the transactions that contain them. If it receives a rollback, it discards the operations for that transaction. If it receives a commit, it persists the transaction to disk in a

series of files called a trail, where it is queued for propagation to the target system. All the operations in each transaction are written to the trail and are in the order in which they were committed to the source technology. This design ensures both speed and data integrity. The format of the data written to trail files depends on the source technology.

- [Add Extract](#)
- [Amazon MSK](#)
- [Amazon DocumentDB](#)
To capture messages from Amazon DocumentDB and convert them into logical change records using Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA), you can utilize MongoDB Extract.
- [Apache Cassandra](#)
The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) capture (Extract) for Cassandra Extract is used to get changes from Apache Cassandra databases.
- [Apache Kafka](#)
The Oracle GoldenGate capture (Extract) for Kafka is used to read messages from a Kafka topic or topics and convert data into logical change records written to GoldenGate trail files. This section explains how to use Oracle GoldenGate capture for Kafka.
- [Azure Event Hubs](#)
- [Confluent Kafka](#)
- [DataStax](#)
- [Java Message Service \(JMS\)](#)
- [MongoDB](#)
The Oracle GoldenGate capture (Extract) for MongoDB is used to get changes from MongoDB databases.
- [Microsoft Fabric Eventstreams](#)
This chapter describes how to use the Microsoft Fabric eventstreams .
- [OCI Streaming](#)
- [Setting Up and Running the Kinesis Streams Handler](#)

10.1.13 Setting Up and Running the Kinesis Streams Handler

Instructions for configuring the Kinesis Streams Handler components and running the handler are described in the following sections.

Use the following steps to set up the Kinesis Streams Handler:

1. Create an Amazon AWS account at <https://aws.amazon.com/>.
2. Log into Amazon AWS.
3. From the main page, select **Kinesis** (under the Analytics subsection).
4. Select Amazon Kinesis Streams **Go to Streams** to create Amazon Kinesis streams and shards within streams.
5. Create a client ID and secret to access Kinesis.
The Kinesis Streams Handler requires these credentials at runtime to successfully connect to Kinesis.
6. Create the client ID and secret:
 - a. Select your name in AWS (upper right), and then in the list select **My Security Credentials**.

- b. Select **Access Keys** to create and manage access keys.
Note your client ID and secret upon creation.

The client ID and secret can only be accessed upon creation. If lost, you have to delete the access key, and then recreate it.

- [Set the Classpath in Kinesis Streams Handler](#)
- [Kinesis Streams Handler Configuration](#)
- [Using Templates to Resolve the Stream Name and Partition Name](#)
- [Resolving AWS Credentials](#)
- [Configuring the Proxy Server for Kinesis Streams Handler](#)
- [Configuring Security in Kinesis Streams Handler](#)

10.1.13.1 Set the Classpath in Kinesis Streams Handler

You must configure the `gg.classpath` property in the Java Adapter properties file to specify the JARs for the AWS Kinesis Java SDK as follows:

```
gg.classpath= {download_dir}/aws-java-sdk-2.28.11/lib/*:{download_dir} /aws-java-sdk-2.28.11/third-party/lib/*
```

10.1.13.2 Kinesis Streams Handler Configuration

You configure the Kinesis Streams Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the Kinesis Streams Handler, you must first configure the handler type by specifying `gg.handler.name.type=kinesis_streams` and the other Kinesis Streams properties as follows:

Table 10-1 Kinesis Streams Handler Configuration Properties

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.handler.name.type</code>	Required	<code>kinesis_streams</code>	None	Selects the Kinesis Streams Handler for streaming change data capture into Kinesis.
<code>gg.handler.name.mode</code>	Optional	<code>op</code> or <code>tx</code>	<code>op</code>	Choose the operating mode.
<code>gg.handler.name.region</code>	Required	The Amazon region name which is hosting your Kinesis instance.	None	Setting of the Amazon AWS region name is required.
<code>gg.handler.name.proxyServer</code>	Optional	The host name of the proxy server.	None	Set the host name of the proxy server if connectivity to AWS is required to go through a proxy server.

Table 10-1 (Cont.) Kinesis Streams Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.proxyPort</code>	Optional	The port number of the proxy server.	None	Set the port name of the proxy server if connectivity to AWS is required to go through a proxy server.
<code>gg.handler.name.proxyUsername</code>	Optional	The username of the proxy server (if credentials are required).	None	Set the username of the proxy server if connectivity to AWS is required to go through a proxy server and the proxy server requires credentials.
<code>gg.handler.name.proxyPassword</code>	Optional	The password of the proxy server (if credentials are required).	None	Set the password of the proxy server if connectivity to AWS is required to go through a proxy server and the proxy server requires credentials.
<code>gg.handler.name.deferFlushAtTxCommit</code>	Optional	<code>true false</code>	<code>false</code>	When set to false, the Kinesis Streams Handler will flush data to Kinesis at transaction commit for write durability. However, it may be preferable to defer the flush beyond the transaction commit for performance purposes, see Kinesis Handler Performance Considerations .

Table 10-1 (Cont.) Kinesis Streams Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.deferFlushOpCo</code> <code>unt</code>	Optional	Integer	None	Only applicable if <code>gg.handler.name</code> <code>.deferFlushAtTx</code> Commit is set to true. This parameter marks the minimum number of operations that must be received before triggering a flush to Kinesis. Once this number of operations are received, a flush will occur on the next transaction commit and all outstanding operations will be moved from the Kinesis Streams Handler to AWS Kinesis.
<code>gg.handler.name</code> <code>.formatPerOp</code>	Optional	true false	true	When set to true, it will send messages to Kinesis, once per operation (insert, delete, update). When set to false, operations messages will be concatenated for all the operations and a single message will be sent at the transaction level. Kinesis has a limitation of 1MB max message size. If 1MB is exceeded then transaction level message will be broken up into multiple messages.

Table 10-1 (Cont.) Kinesis Streams Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.customMessageGroup</code>	Optional	<code>oracle.goldengate.handler.kinesis.KinesisJsonTxMessageGroup</code>	None	This configuration parameter provides the ability to group Kinesis messages using custom logic. Only one implementation is included in the distribution at this time. The <code>oracle.goldengate.handler.kinesis.KinesisJsonTxMessageGroup</code> is a custom message which groups JSON operation messages representing operations into a wrapper JSON message that encompasses the transaction. Setting of this value overrides the setting of the <code>gg.handler.formatPerOp</code> setting. Using this feature assumes that the customer is using the JSON formatter (that is <code>gg.handler.name.format=json</code>).
<code>gg.handler.name.streamMappingTemplate</code>	Required	A template string value to resolve the Kinesis message partition key (message key) at runtime.	None	See Using Templates to Resolve the Stream Name and Partition Name for more information.
<code>gg.handler.name.partitionMappingTemplate</code>	Required	A template string value to resolve the Kinesis message partition key (message key) at runtime.	None	See Using Templates to Resolve the Stream Name and Partition Name for more information.

Table 10-1 (Cont.) Kinesis Streams Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.format</code>	Required	Any supported pluggable formatter.	<code>delimitedtext json json_row xml avro_row avro_opt</code>	Selects the operations message formatter. JSON is likely the best fit for Kinesis.
<code>gg.handler.name.enableStreamCreation</code>	Optional	<code>true</code>	<code>true false</code>	By default, the Kinesis Handler automatically creates Kinesis streams if they do not already exist. Set to <code>false</code> to disable to automatic creation of Kinesis streams.
<code>gg.handler.name.shardCount</code>	Optional	Positive integer.	<code>1</code>	A Kinesis stream contains one or more shards. Controls the number of shards on Kinesis streams that the Kinesis Handler creates. Multiple shards can help improve the ingest performance to a Kinesis stream. Use only when <code>gg.handler.name.enableStreamCreation</code> is set to <code>true</code> .
<code>gg.handler.name.proxyProtocol</code>	Optional	<code>HTTP HTTPS</code>	<code>HTTP</code>	Sets the proxy protocol connection to the proxy server for additional level of security. The client first performs an SSL handshake with the proxy server, and then an SSL handshake with Amazon AWS. This feature was added into the Amazon SDK in version 1.11.396 so you must use at least that version to use this property.

Table 10-1 (Cont.) Kinesis Streams Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.enableSTS</code>	Optional	true false	false	Set to true, to enable the Kinesis Handler to access Kinesis credentials from the AWS Security Token Service. Ensure that the AWS Security Token Service is enabled if you set this property to true.
<code>gg.handler.name.STSRegion</code>	Optional	Any legal AWS region specifier.	The region is obtained from the <code>gg.handler.name.region</code> property.	Use to resolve the region for the STS call. It's only valid if the <code>gg.handler.name.enableSTS</code> property is set to true. You can set a different AWS region for resolving credentials from STS than the configured Kinesis region.
<code>gg.handler.name.kinesis.accessKeyId</code>	Optional	A valid AWS access key.	None	Set this parameter to explicitly set the access key for AWS. This parameter has no effect if <code>gg.handler.name.enableSTS</code> is set to true. If unset, credentials resolution falls back to the AWS default credentials provider chain. Optionally, you can configure the session token (<code>gg.handler.kinesis.sessionToken</code>), which indicates temporary credentials. The access key and secret key MUST be set for the session token configuration to be valid.

Table 10-1 (Cont.) Kinesis Streams Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.kinesis.secretKey</code>	Optional	A valid AWS secret key.	None	Set this parameter to explicitly set the secret key for AWS. This parameter has no effect if <code>gg.handler.name.enableSTS</code> is set to <code>true</code> . If unset, credentials resolution falls back to the AWS default credentials provider chain. Optionally, you can configure the session token (<code>gg.handler.kinesis.sessionToken</code>), which indicates temporary credentials. The access key and secret key MUST be set for the session token configuration to be valid.

10.1.13.3 Using Templates to Resolve the Stream Name and Partition Name

The Kinesis Streams Handler provides the functionality to resolve the stream name and the partition key at runtime using a template configuration value. Templates allow you to configure static values and keywords. Keywords are used to dynamically replace the keyword with the context of the current processing. Templates are applicable to the following configuration parameters:

```
gg.handler.name.streamMappingTemplate
gg.handler.name.partitionMappingTemplate
```

Source database transactions are made up of 1 or more individual operations which are the individual inserts, updates, and deletes. The Kinesis Handler can be configured to send one message per operation (insert, update, delete). Alternatively, it can be configured to group operations into messages at the transaction level. Many of the template keywords resolve data based on the context of an individual source database operation. Therefore, many of the keywords *do not work* when sending messages at the transaction level. For example `${fullyQualifiedTableName}` does not work when sending messages at the transaction level. The `gg.handler.kinesis.fullyQualifiedTableName` property resolves to the qualified source table name for an operation. Transactions can contain multiple operations for many source tables. Resolving the fully-qualified table name for messages at the transaction level is non-deterministic and so abends at runtime.

For more information about the Template Keywords, see [Template Keywords](#).

Example Templates

The following describes example template configuration values and the resolved values.

Example Template	Resolved Value
<code>\${groupName}_\${fullyQualifiedTableName}</code>	KINESIS001_DB0.TABLE1
<code>prefix_\${schemaName}_\${tableName}_suffix</code>	prefix_DB0_TABLE1_suffix
<code>\${currentTimestamp[yyyy-mm-dd hh:MM:ss.SSS]}</code>	2017-05-17 11:45:34.254

10.1.13.4 Resolving AWS Credentials

- [AWS Kinesis Client Authentication](#)
The Kinesis Handler is a client connection to the AWS Kinesis cloud service. The AWS cloud must be able to successfully authenticate the AWS client in order in order to successfully interface with Kinesis.

10.1.13.4.1 AWS Kinesis Client Authentication

The Kinesis Handler is a client connection to the AWS Kinesis cloud service. The AWS cloud must be able to successfully authenticate the AWS client in order in order to successfully interface with Kinesis.

The AWS client authentication has become increasingly complicated as more authentication options have been added to the Kinesis Stream Handler. This topic explores the different use cases for AWS client authentication.

- [Explicit Configuration of the Client ID and Secret](#)
A client ID and secret are generally the required credentials for the Kinesis Handler to interact with Amazon Kinesis. A client ID and secret are generated using the Amazon AWS website.
- [Use of the AWS Default Credentials Provider Chain](#)
If the `gg.eventhandler.name.accessKeyId` and `gg.eventhandler.name.secretKey` are unset, then credentials resolution reverts to the AWS default credentials provider chain. The AWS default credentials provider chain provides various ways by which the AWS credentials can be resolved.
- [AWS Federated Login](#)
The use case is when you have your on-premise system login integrated with AWS. This means that when you log into an on-premise machine, you are also logged into AWS.

10.1.13.4.1.1 Explicit Configuration of the Client ID and Secret

A client ID and secret are generally the required credentials for the Kinesis Handler to interact with Amazon Kinesis. A client ID and secret are generated using the Amazon AWS website.

These credentials can be explicitly configured in the Java Adapter Properties file as follows:

```
gg.handler.name.accessKeyId=
gg.handler.name.secretKey=
```

Furthermore, the Oracle Wallet functionality can be used to encrypt these credentials.

10.1.13.4.1.2 Use of the AWS Default Credentials Provider Chain

If the `gg.eventhandler.name.accessKeyId` and `gg.eventhandler.name.secretKey` are unset, then credentials resolution reverts to the AWS default credentials provider chain. The AWS default credentials provider chain provides various ways by which the AWS credentials can be resolved.

For more information about the default credential provider chain and order of operations for AWS credentials resolution, see [Working with AWS Credentials](#).

When Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) runs on an AWS Elastic Compute Cloud (EC2) instance, the general use case is to resolve the credentials from the EC2 metadata service. The AWS default credentials provider chain provides resolution of credentials from the EC2 metadata service as one of the options.

10.1.13.4.1.3 AWS Federated Login

The use case is when you have your on-premise system login integrated with AWS. This means that when you log into an on-premise machine, you are also logged into AWS.

In this use case:

- You may not want to generate client IDs and secrets. (Some users disable this feature in the AWS portal).
- The client AWS applications need to interact with the AWS Security Token Service (STS) to obtain an authentication token for programmatic calls made to Kinesis.

This feature is enabled by setting the following: `gg.eventhandler.name.enableSTS=true`.

10.1.13.5 Configuring the Proxy Server for Kinesis Streams Handler

Oracle GoldenGate can be used with a proxy server using the following parameters to enable the proxy server:

```
gg.handler.name.proxyServer=  
gg.handler.name.proxyPort=80  
gg.handler.name.proxyUsername=username  
gg.handler.name.proxyPassword=password
```

Sample configurations:

```
gg.handlerlist=kinesis  
gg.handler.kinesis.type=kinesis_streams  
gg.handler.kinesis.mode=op  
gg.handler.kinesis.format=json  
gg.handler.kinesis.region=us-west-2  
gg.handler.kinesis.partitionMappingTemplate=TestPartitionName  
gg.handler.kinesis.streamMappingTemplate=TestStream  
gg.handler.kinesis.deferFlushAtTxCommit=true  
gg.handler.kinesis.deferFlushOpCount=1000  
gg.handler.kinesis.formatPerOp=true  
#gg.handler.kinesis.customMessageGrouper=oracle.goldengate.handler.kinesis.KinesisJsonTxMessageGrouper  
gg.handler.kinesis.proxyServer=www-proxy.myhost.com  
gg.handler.kinesis.proxyPort=80
```

10.1.13.6 Configuring Security in Kinesis Streams Handler

The Amazon Web Services (AWS) Kinesis Java SDK uses HTTPS to communicate with Kinesis. Mutual authentication is enabled. The AWS server passes a Certificate Authority (CA) signed certificate to the AWS client which allow the client to authenticate the server. The AWS client passes credentials (client ID and secret) to the AWS server which allows the server to authenticate the client.

10.1.1 Add Extract

Extract process can be configured in GoldenGate for Distributed Applications and Analytics (GG for DAA) 26ai web-based user interface.

To create an extract:

1. Complete prerequisites based on source technology type. See the following sections for technology-specific details.
2. Download dependencies using dependency downloader for your source technology and note the path to dependency files.
3. If required, create the credential store entry. See the following sections for technology-specific details.
4. In GG for DAA UI, go to **Administration Service** and click **Add Extract**.
5. In **Extract Information**, provide a Process Name for the extract, select source technology and extract type.
6. In **Extract Options**, provide a name for Extract Trail, select Source Credentials and Alias.
7. Managed Options are optional, enable if needed.

Option	Description
Profile Name	Provides the name of the autostart and autorestart profile. You can select the default or custom options. If you have already created a profile, then you can select that profile also. If you select the Custom option, then you can set up a new profile from this section itself.
Critical to deployment health	(Oracle only) Enable this option if the profile is critical for the deployment health. <i>Note</i> This option only appears while creating the Extract or Replicat and not when you set up the managed processes in the Profiles page.
Auto Start	Enables autostart for the process
Startup Delay	Time to wait in seconds before starting the process
Auto Restart	Configures how to restart the process if it terminates

Option	Description
Max Retries	Specify the maximum number of retries to try to start the process
Retry Delay	Delay time in trying to start the process
Retries Window	The duration interval to try to start the process
Restart on Failure only	If true, the task is only restarted if it fails.
Disable Task After Retries Exhausted	If true, then the task is disabled after exhausting all attempts to restart the process.

Parameter File will be populated based on your source technology selection. In Parameter File, update the fields marked as TODO.

8. Create and Run will register the extract and start it.

10.1.2 Amazon MSK

To capture messages from Amazon MSK and parse into logical change records with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA), you can use Kafka Extract. For more information, see [Apache Kafka](#) as source.

10.1.3 Amazon DocumentDB

To capture messages from Amazon DocumentDB and convert them into logical change records using Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA), you can utilize MongoDB Extract.

There are several key distinctions between MongoDB capture and Amazon DocumentDB capture:

- Amazon DocumentDB does not support Bidirectional Replication.
- Unlike MongoDB, Amazon DocumentDB does not have an operations log (oplog). To read events in Amazon DocumentDB, you must explicitly enable change streams feature for a specific collection by using the add trandata feature.
- Change streams in Amazon DocumentDB are retained for a default period of 3 hours after the event is recorded. This retention can be extended up to a maximum of 7 days by modifying the `change_stream_log_retention_duration` parameter.

Note

- For more information, see [MongoDB](#) as a source.

For information on *How to Connect to an Amazon DocumentDB Cluster from outside an Amazon VPC*, see the [Amazon DocumentDB](#) documentation.

10.1.4 Apache Cassandra

The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) capture (Extract) for Cassandra Extract is used to get changes from Apache Cassandra databases.

You can select Extract starting positions from UI from **Extract Options**, under **Begin**. You can either select **Now** or define a Custom Time.

This chapter describes how to use the GG for DAA Capture for Cassandra Extract.

- [Overview](#)
- [Setting Up Cassandra Extract Change Data Capture](#)
- [Deduplication](#)
- [Topology Changes](#)
- [Data Availability in the CDC Logs](#)
- [Using Initial Load Extract](#)
- [Using Change Data Capture Extract](#)
- [Replicating to RDMBS Targets](#)
- [Partition Update or Insert of Static Columns](#)
- [Partition Delete](#)
- [Security and Authentication](#)
- [Cleanup of CDC Commit Log Files](#)
You can use the Cassandra CDC commit log purger program to purge the CDC commit log files that are not in use.
- [Multiple Extract Support](#)
- [CDC Configuration Reference](#)
- [Troubleshooting](#)
- [Cassandra Capture Client Dependencies](#)
What are the dependencies for the Cassandra Capture (Extract) to connect to Apache Cassandra databases?

10.1.4.1 Overview

Apache Cassandra is a NoSQL Database Management System designed to store large amounts of data. A Cassandra cluster configuration provides horizontal scaling and replication of data across multiple machines. It can provide high availability and eliminate a single point of failure by replicating data to multiple nodes within a Cassandra cluster. Apache Cassandra is open source and designed to run on low-cost commodity hardware.

Cassandra relaxes the axioms of a traditional relational database management systems (RDBMS) regarding atomicity, consistency, isolation, and durability. When considering implementing Cassandra, it is important to understand its differences from a traditional RDBMS and how those differences affect your specific use case.

Cassandra provides eventual consistency. Under the eventual consistency model, accessing the state of data for a specific row eventually returns the latest state of the data for that row as defined by the most recent change. However, there may be a latency period between the creation and modification of the state of a row and what is returned when the state of that row is queried. The benefit of eventual consistency is that the latency period is predicted based on your Cassandra configuration and the level of work load that your Cassandra cluster is currently under, see <http://cassandra.apache.org/>.

Review the data type support, see [About the Cassandra Data Types](#).

10.1.4.2 Setting Up Cassandra Extract Change Data Capture

Prerequisites

- Apache Cassandra cluster must have at least one node up and running.

- Read and write access to CDC commit log files on every live node in the cluster is done through SFTP or NFS. For more information, see [Setup SSH Connection to the Cassandra Nodes](#).
- Every node in the Cassandra cluster must have the `cdc_enabled` parameter set to `true` in the `cassandra.yaml` configuration file.
- Virtual nodes must be enabled on every Cassandra node by setting the `num_tokens` parameter in `cassandra.yaml`.
- You must download the third party libraries using Dependency downloader scripts. For more information, see [Cassandra Capture Client Dependencies](#).
- New tables can be created with Change Data Capture (CDC) enabled using the `WITH CDC=true` clause in the `CREATE TABLE` command. For example:

```
CREATE TABLE ks_demo_rep1.mytable (col1 int, col2 text, col3 text, col4 text,
PRIMARY KEY (col1)) WITH cdc=true;
```

You can enable CDC on existing tables as follows:

```
ALTER TABLE ks_demo_rep1.mytable WITH cdc=true;
```

- [Setup SSH Connection to the Cassandra Nodes](#)
Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) transfers Cassandra commit log files from all the Cassandra nodes. To allow Oracle GoldenGate to transfer commit log files using secure shell protocol (SFTP), generate a `known_hosts` SSH file.
- [Data Types](#)
- [Cassandra Database Operations](#)
- [Set up Credential Store Entry to Detect Source Type](#)

10.1.4.2.1 Setup SSH Connection to the Cassandra Nodes

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) transfers Cassandra commit log files from all the Cassandra nodes. To allow Oracle GoldenGate to transfer commit log files using secure shell protocol (SFTP), generate a `known_hosts` SSH file.

To generate a `known_hosts` SSH file:

1. Create a text file with all the Cassandra node addresses, one per line. For example:

```
cat nodes.tx
10.1.1.1
10.1.1.2
10.1.1.3
```
2. Generate the `known_hosts` file as follows: `ssh-keyscan -t rsa -f nodes.txt >> known_hosts`
3. Edit the extract parameter file to include this configuration: `TRANLOGOPTIONS SFTP KNOWNHOSTSFILE /path/to/ssh/known_hosts`.

10.1.4.2.2 Data Types

Supported Cassandra Source Data Types

The following are the supported source data types:

- ASCII

- BIGINT
- BLOB
- BOOLEAN
- DATE
- DECIMAL
- DOUBLE
- DURATION
- FLOAT
- INET
- INT
- SMALLINT
- TEXT
- TIME
- TIMESTAMP
- TIMEUUID
- TINYINT
- UUID
- VARCHAR
- VARINT

Unsupported Source Data Types

The following are the unsupported source data types:

- COUNTER
- MAP
- SET
- LIST
- UDT (user defined type)
- TUPLE
- CUSTOM_TYPE

10.1.4.2.3 Cassandra Database Operations

Supported Operations

The following are the supported operations:

- INSERT
- UPDATE (Captured as INSERT)
- DELETE

Unsupported Operations

The TRUNCATE DDL (CREATE, ALTER, and DROP) operation is not supported. Because the Cassandra commit log files do not record any before images for the UPDATE or DELETE operations. The result is that the captured operations can never have a before image. Features that rely on before image records, such as Conflict Detection and Resolution, are not available.

10.1.4.2.4 Set up Credential Store Entry to Detect Source Type

The database type for capture is based on the prefix in the database credential `userid`. The generic format for `userid` is as follows: `<dbtype>://<db-user>@<comma separated list of server addresses>:<port>`

The `userid` can have multiple server/nodes addresses.

More than one node address can be configured in the `userid`.

In the Administration Service, you can create the credential store entry under DB Connections. To add Trandata, go to DB Connections in Administration Service, connect to your database from credential entry and Add Trandata.

Example

```
alter credentialstore add user cassandra://db-user@127.0.0.1,127.0.0.2:9042 password db-passwd alias cass
```

10.1.4.3 Deduplication

One of the features of a Cassandra cluster is its high availability. To support high availability, multiple redundant copies of table data are stored on different nodes in the cluster. Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) Cassandra Capture automatically filters out duplicate rows (**deduplicate**). Deduplication is active by default. Oracle recommends using it if your data is captured and applied to targets where duplicate records are discouraged (for example RDBMS targets).

10.1.4.4 Topology Changes

Cassandra nodes can change their status (**topology change**) and the cluster can still be alive. GG for DAA Cassandra Capture can detect the node status changes and react to these changes when applicable. The Cassandra capture process can detect the following events happening in the cluster:

- Node shutdown and boot.
- Node decommission and commission.
- New keyspace and table created.

Due to topology changes, if the capture process detects that an active producer node goes down, it tries to recover any missing rows from an available replica node. During this process, there is a possibility of data duplication for some rows. This is a transient data duplication due to the topology change. For more details about reacting to changes in topology, see [Troubleshooting](#).

10.1.4.5 Data Availability in the CDC Logs

The Cassandra CDC API can only read data from commit log files in the CDC directory. There is a latency for the data in the active commit log directory to be archived (moved) to the CDC commit log directory.

The input data source for the Cassandra capture process is the CDC commit log directory. There could be delays for the data to be captured mainly due to the commit log files not yet visible to the capture process.

On a production cluster with a lot of activity, this latency is very minimal as the data is archived from the active commit log directory to the CDC commit log directory in the order of microseconds.

10.1.4.6 Using Initial Load Extract

Cassandra Extract supports the standard initial load capability to extract source table data to GG for DAA trail files.

Initial load for Cassandra can be performed to synchronize tables, either as a prerequisite step to replicating changes or as a standalone function.

Direct loading from a source Cassandra table to any target table is *not* supported.

10.1.4.7 Using Change Data Capture Extract

Review the example .prm files from GG for DAA installation directory under \$HOME/AdapterExamples/big-data/cassandrapture.

1. In **Administration Service**, click **Add Extract**.
2. Enter a **Process Name**, select **Source** as Cassandra and select **Change Data Capture Extract**.
3. Provide a name for Extract Trail and select **Source Credentials** you created for Cassandra.
4. Update **Managed Options** if necessary.
5. In the **Parameter File**, update the fields marked as **TODO**.
6. **Create & Run**.
7. Configure the Extract parameter file:

Apache Cassandra 4x SDK, compatible with Apache Cassandra 4.0 version
Extract parameter file:

```
-- ggsci> alter credentialstore add user cassandra://db-user@127.0.0.1 password db-
passwd alias cass
EXTRACT groupname
```

```
JVMOPTIONS CLASSPATH ggjava/ggjava.jar:DependencyDownloader/dependencies/
cassandra_capture_4x/*
JVMOPTIONS BOOTOPTIONS -Dcassandra.config=file://{path/to/apache-cassandra-4.x}/
config/cassandra.yaml -Dcassandra.datacenter={datacenter-name}
```

```
TRANLOGOPTIONS CDCREADERSDKVERSION 4x
TRANLOGOPTIONS CDCLOGDIRTEMPLATE /path/to/data/cdc_raw
SOURCEDB USERIDALIAS cass
EXTTRAIL trailprefix
TABLE source.*;
```

- a. Provide the cassandra.yaml file path using JVMOPTIONS BOOTOPTIONS.

```
JVMOPTIONS BOOTOPTIONS -Dcassandra.config=file://{path/to/apache-  
cassandra-4.x}/config/cassandra.yaml -Dcassandra.datacenter={datacenter-name}
```

Note

For a remote capture (when the Cassandra server is not on the same machine as Oracle GoldenGate), you need to copy a sample `cassandra.yaml` for your Apache Cassandra version onto your GoldenGate Machine and use this path for this `cassandra.yaml` file path configuration.

- b. Configure `cassandra` datacenter name under `JVMOPTIONS BOOTOPTIONS`. If you do not provide a value, then by default, `datacenter1` is considered.

Apache Cassandra 3x SDK, compatible with Apache Cassandra 3.9, 3.10, 3.11

Extract parameter file:

```
-- ggsci> alter credentialstore add user cassandra://db-user@127.0.0.1 password db-  
passwd alias cass  
JVMOPTIONS CLASSPATH ggjava/ggjava.jar:DependencyDownloader/dependencies/  
cassandra_capture_3x/*  
TRANLOGOPTIONS CDCREADERSDKVERSION 3x  
TRANLOGOPTIONS CDCLOGDIRTEMPLATE /path/to/data/cdc_raw  
SOURCEDB USERIDALIAS cass  
EXTTRAIL trailprefix  
TABLE source.*;
```

DSE Cassandra SDK, compatible with DSE Cassandra 6.x versions

Extract parameter file

```
-- ggsci> alter credentialstore add user cassandra://db-user@127.0.0.1 password  
db-passwd alias cass  
EXTRACT groupname  
JVMOPTIONS CLASSPATH ggjava/ggjava.jar:{/path/to/dse-6.x}/resources/  
cassandra/lib/*:{/path/to/dse-6.x}/lib/*:{/path/to/dse-6.x}/resources/dse/lib/  
*:DependencyDownloader/dependencies/cassandra_capture_dse/*  
JVMOPTIONS BOOTOPTIONS -Dcassandra.config=file://{path/to/dse-6.x}/resources/  
cassandra/conf/cassandra.yaml -Dcassandra.datacenter={datacenter-name}  
TRANLOGOPTIONS CDCREADERSDKVERSION dse  
TRANLOGOPTIONS CDCLOGDIRTEMPLATE /path/to/data/cdc_raw  
SOURCEDB USERIDALIAS cass  
EXTTRAIL trailprefix  
TABLE source.*;
```

- a. Provide the `cassandra.yaml` file path using `JVMOPTIONS BOOTOPTIONS`:

```
JVMOPTIONS BOOTOPTIONS -Dcassandra.config=file://{path/to/dse-6.x}/resources/  
cassandra/conf/cassandra.yaml -Dcassandra.datacenter={datacenter-name}
```

Note

For a remote capture (when the Cassandra server is not on the same machine as Oracle GoldenGate), you need to copy a sample `cassandra.yaml` for your DSE version onto your GoldenGate Machine and use this path for this `cassandra.yaml` file path configuration.

- b. Configure cassandra datacenter name under JVMOPTIONS B00T0PTIONS. If you do not provide a value, then by default, Cassandra is considered.

Note

For DSE 5.x version, configure the extract with Apache 3x SDK as explained in the Apache 3x section.

- [Handling Schema Evolution](#)

10.1.4.7.1 Handling Schema Evolution

Syntax

```
TRANLOGOPTIONS TRACKSCHEMACHANGES
```

This will enable extract to capture table level DDL changes from the source at runtime.

Enable this to ensure that the table metadata within the trail stays in sync with the source without any downtime.

When TRACKSCHEMACHANGES is disabled, the capture process will ABEND if a DDL change is detected at the source table.

Note

This feature is disabled by default. To enable, update the extract prn file as shown in the syntax above.

10.1.4.8 Replicating to RDMBS Targets

You must take additional care when replicating source UPDATE operations from Cassandra trail files to RDMBS targets. Any source UPDATE operation appears as an INSERT record in the Oracle GoldenGate trail file. Replicat may abend when a source UPDATE operation is applied as an INSERT operation on the target database.

You have these options:

- **OVERRIDEDUPS:** If you expect that the source database is to contain mostly INSERT operations and very few UPDATE operations, then OVERRIDEDUPS is the recommended option. Replicat can recover from duplicate key errors while replicating the small number of the source UPDATE operations.
- **UPDATEINSERTS and INSERTMISSINGUPDATES:** Use this configuration if the source database is expected to contain mostly UPDATE operations and very few INSERT operations. With this configuration, Replicat has fewer missing row errors to recover, which leads to better throughput.
- No additional configuration is required if the target table can accept duplicate rows or you want to abend Replicat on duplicate rows.

If you configure Replicat to use BATCHSQL, then there may be duplicate row or missing row errors in batch mode. Although there is a reduction in the Replicat throughput due to these errors, Replicat automatically recovers from these errors. If the source operations are mostly INSERTS, then BATCHSQL is a good option.

10.1.4.9 Partition Update or Insert of Static Columns

When the source Cassandra table has static columns, the static column values can be modified by skipping any clustering key columns that are in the table.

For example:

```
create table ks_demo_rep1.nls_staticcol
(
    teamname text,
    manager text static,
    location text static,
    membername text,
    nationality text,
    position text,
    PRIMARY KEY ((teamname), membername)
)
WITH cdc=true;
insert into ks_demo_rep1.nls_staticcol (teamname, manager, location) VALUES
('Red Bull', 'Christian Horner', '<unknown>')
```

The insert CQL is missing the clustering key membername. Such an operation is a partition insert.

Similarly, you could also update a static column with just the partition keys in the WHERE clause of the CQL that is a partition update operation. Cassandra Extract cannot write a INSERT or UPDATE operation into the trail with missing key columns. It abends on detecting a partition INSERT or UPDATE operation.

10.1.4.10 Partition Delete

A Cassandra table may have a primary key composed on one or more partition key columns and clustering key columns. When a DELETE operation is performed on a Cassandra table by skipping the clustering key columns from the WHERE clause, it results in a partition delete operation.

For example:

```
create table ks_demo_rep1.table1
(
    col1 ascii, col2 bigint, col3 boolean, col4 int,
    PRIMARY KEY((col1, col2), col4)
) with cdc=true;

delete from ks_demo_rep1.table1 where col1 = 'asciival' and col2 =
9876543210; /** skipped clustering key column col4 **/
```

Cassandra Extract cannot write a DELETE operation into the trail with missing key columns and abends on detecting a partition DELETE operation.

10.1.4.11 Security and Authentication

- Cassandra Extract can connect to a Cassandra cluster using username and password based authentication and SSL authentication.

- Connection to Kerberos enabled Cassandra clusters is *not* supported in this release.
- [Configuring SSL](#)

10.1.4.11.1 Configuring SSL

To enable SSL, add the SSL parameter to your GLOBALS file or Extract parameter file. Additionally, a separate configuration is required for the Java and CPP drivers, see [CDC Configuration Reference](#).

SSL configuration for Java driver (GLOBALS file)

```
JVMB00TOPTIONS -Djavax.net.ssl.trustStore=/path/to/SSL/truststore.file
-Djavax.net.ssl.trustStorePassword=password
-Djavax.net.ssl.keyStore=/path/to/SSL/keystore.file
-Djavax.net.ssl.keyStorePassword=password
```

SSL configuration for Java driver (Extract parameter file)

You can also configure the SSL parameters in the Extract parameter file as follows:

```
JVMOPTIONS BOOTOPTIONS -Djavax.net.ssl.trustStore=/path/to/SSL/truststore.file
-Djavax.net.ssl.trustStorePassword=password
-Djavax.net.ssl.keyStore=/path/to/SSL/keystore.file
-Djavax.net.ssl.keyStorePassword=password
```

Note

The Extract parameter file configuration has a higher precedence.

The keystore and truststore certificates can be generated using these instructions:

<https://docs.datastax.com/en/cassandra/3.0/cassandra/configuration/secureSSLIntro.html>

Using Apache Cassandra 4x SDK / DSE Cassandra SDK

To configure SSL while capturing from Apache Cassandra 4.x versions or DSE Cassandra 6.x versions, do the following:

1. Create the application.conf file with the following properties and override with appropriate values :

```
datastax-java-driver {
  advanced.ssl-engine-factory {
    class = DefaultSslEngineFactory

    # Whether or not to require validation that the hostname of the server
    certificate's common
    # name matches the hostname of the server being connected to. If not set,
    defaults to true.
    hostname-validation = false

    # The locations and passwords used to access truststore and keystore contents.
    # These properties are optional. If either truststore-path or keystore-path are
    specified,
    # the driver builds an SSLContext from these files. If neither option is
    specified, the
    # default SSLContext is used, which is based on system property configuration.
    truststore-path = {path to truststore file}
    truststore-password = password
```



```

        keystore-path = {path to keystore file}
        keystore-password = cassandra
    }
}

```

2. Provide path of the directory containing the application.conf file under JVMCLASSPATH as follows:

```

JVMCLASSPATH
ggjava/ggjava.jar:DependencyDownloader/dependencies/cassandra_capture_4x/*:/path/to/
driver/config

```

Note

This is valid only in case of the GLOBALS file.

You can also configure the SSL parameters in the Extract parameter file as follows:

```

JVMOPTIONS CLASSPATH
ggjava/ggjava.jar:DependencyDownloader/dependencies/cassandra_capture_4x/*:/path/to/
driver/config/

```

For more information, see <https://github.com/datastax/java-driver/blob/4.x/core/src/main/resources/reference.conf>.

SSL configuration for Cassandra CPP driver

To operate with an SSL configuration, you have to add the following parameter in the Oracle GoldenGate GLOBALS file or Extract parameter file:

```

CPPDRIVEROPTIONS SSL PEMPUBLICKEYFILE /path/to/PEM/formatted/public/key/file/
cassandra.pem CPPDRIVEROPTIONS SSL PEERCERTVERIFICATIONFLAG 0

```

This configuration is required to connect to a Cassandra cluster with SSL enabled. Additionally, you need to add these settings to your cassandra.yaml file:

```

client_encryption_options:
  enabled: true
  # If enabled and optional is set to true encrypted and unencrypted connections are
  handled.
  optional: false
  keystore: /path/to/keystore
  keystore_password: password
  require_client_auth: false

```

The PEM formatted certificates can be generated using these instructions:

<https://docs.datastax.com/en/developer/cpp-driver/2.8/topics/security/ssl/>

10.1.4.12 Cleanup of CDC Commit Log Files

You can use the Cassandra CDC commit log purger program to purge the CDC commit log files that are not in use.

For more information, see [How to Run the Purge Utility](#).

- [Cassandra CDC Commit Log Purger](#)
A purge utility for Cassandra Handler to purge the staged CDC commit log files. Cassandra Extract moves the CDC commit log files (located at \$CASSANDRA/data/cdc_raw) on each node to a staging directory for processing.

10.1.4.12.1 Cassandra CDC Commit Log Purger

A purge utility for Cassandra Handler to purge the staged CDC commit log files. Cassandra Extract moves the CDC commit log files (located at \$CASSANDRA/data/cdc_raw) on each node to a staging directory for processing.

For example, if the cdc_raw commit log directory is /path/to/cassandra/home/data/cdc_raw, the staging directory is /path/to/cassandra/home/data/cdc_raw/../cdc_raw_staged. The CDC commit log purger purges those files, which are inside cdc_raw_staged based on following logic.

The Purge program scans the oggdir directory for all the following JSON checkpoint files under dirchk/<EXTGRP>_casschk.json. The sample JSON file under dirchk looks similar to the following:

```
{
  "start_timestamp": -1,
  "sequence_id": 34010434,
  "updated_datetime": "2018-04-19 23:24:57.164-0700",
  "nodes": [
    { "address": "10.247.136.146", "offset": 0, "id": 0 }
  ],
  { "address": "10.247.136.142", "file": "CommitLog-6-1524110205398.log",
    "offset": 33554405, "id": 1524110205398 }
  ,
  { "address": "10.248.10.24", "file": "CommitLog-6-1524110205399.log",
    "offset": 33554406, "id": 1524110205399 }
  ]
}
```

For each node address in JSON checkpoint file, the purge program captures the CDC file name and ID. For each ID obtained from the JSON checkpoint file, the purge program looks into the staged CDC commit log directory and purges the commit log files with the id that are lesser then the id captured in JSON file of checkpoint.

Example:

In JSON file, we had ID as 1524110205398.

In CDC Staging directory, we have files as CommitLog-6-1524110205396.log, CommitLog-6-1524110205397.log, and CommitLog-6-1524110205398.log.

The ids derived from CDC staging directory are 1524110205396, 1524110205397 and 1524110205398. The purge utility purges the files in CDC staging directory whose IDs are less than the ID read in JSON file, which is 1524110205398. The files associated with the ID 1524110205396 are 1524110205397 are purged.

- [How to Run the Purge Utility](#)
- [Sample config.properties for Local File System](#)
- [Argument cassCommitLogPurgerConfFile](#)

- [Argument purgeInterval](#)
Setting the optional argument `purgeInterval` helps in configuring the process to run as a daemon.
- [Argument cassUnProcessedFilesPurgeInterval](#)
Setting the optional argument `cassUnProcessedFilesPurgeInterval` helps in purging historical commit logs for all the nodes that do not have a last processed file.

10.1.4.12.1.1 How to Run the Purge Utility

- [Third Party Libraries Needed to Run this Program](#)
- [Command to Run the Program](#)
- [Runtime Arguments](#)

10.1.4.12.1.1.1 Third Party Libraries Needed to Run this Program

```
<dependency>
<groupId>com.jcraft</groupId>
<artifactId>jsch</artifactId>
<version>0.1.54</version>
<scope>provided</scope>
</dependency>
```

10.1.4.12.1.1.2 Command to Run the Program

```
java -Dlog4j.configurationFile=log4j-purge.properties -Dgg.log.level=INFO -cp <OGG_HOME>/
ggjava/resources/lib/*:<OGG_HOME>/thirdparty/cass/jsch-0.1.54.jar
oracle.goldengate.cassandra.commitlogpurger.CassandraCommitLogPurger
--cassCommitLogPurgerConfFile <OGG_HOME>/cassandraPurgeUtil/commitlogpurger.properties
--purgeInterval 1 --cassUnProcessedFilesPurgeInterval 3
```

Where:

- `<OGG_HOME>/ggjava/resources/lib/*` is the directory where the purger utility is located.
- `<OGG_HOME>/thirdparty/cass/jsch-0.1.54.jar` is the dependent jar to execute the purger program.
- `---cassCommitLogPurgerConfFile`, `--purgeInterval` and `--cassUnProcessedFilesPurgeInterval` are run time arguments.

Sample script to run the commit log purger utility:

```
#!/bin/bash
echo "fileSystemType=remote" > commitlogpurger.properties
echo "chkDir=dirchk" >> commitlogpurger.properties
echo "cdcStagingDir=data/cdc_raw_staged" >> commitlogpurger.properties
echo "userName=username" >> commitlogpurger.properties
echo "password=password" >> commitlogpurger.properties
java -cp ogghome/ggjava/resources/lib/*:ogghome/thirdparty/cass/jsch-0.1.54.jar
oracle.goldengate.cassandra.commitlogpurger.CassandraCommitLogPurger
--cassCommitLogPurgerConfFile commitlogpurger.properties
--purgeInterval 1
--cassUnProcessedFilesPurgeInterval 3
```

10.1.4.12.1.1.3 Runtime Arguments

To execute, the utility class `CassandraCommitLogPurger` requires a mandatory run-time argument `cassCommitLogPurgerConfFile`.

Available Runtime arguments to CassandraCommitLogPurger class are:

```
[required] --cassCommitLogPurgerConfFile path to config.properties
[optional] --purgeInterval
[optional] --cassUnProcessedFilesPurgeInterval
```

10.1.4.12.1.2 Sample config.properties for Local File System

```
fileSystemType=local
chkDir=apache-cassandra-3.11.2/data/chkdir/
cdcStagingDir=apache-cassandra-3.11.2/data/$nodeAddress/commitlog/
```

10.1.4.12.1.3 Argument cassCommitLogPurgerConfFile

The required cassCommitLogPurgerConfFile argument takes the config file with following mandate fields.

Table 10-2 Argument cassCommitLogPurgerConfFile

Parameters	Description
fileSystemType	Default: local Mandatory: Yes Legal Values: remote/ local Description: In every live node in the cluster, CDC Staged Commit logs can be accessed through SFTP or NFS. If the fileSystemType is Remote (SFTP) then we need to supply the Host with Port, username, and password/privateKey (with or without passPhase) to connect and do the operations on remote CDC staging directory.
chkDir	Default: None Mandatory: Yes Legal Values: checkpoint directory path Description: Location of Cassandra checkpoint directory where _casschk.json file is located (for example, dirchk/<EXTGRP>_casschk.json).
cdcStagingDir	Default: None Mandatory: Yes Legal Values: staging directory path Description: Location of Cassandra staging directory where CDC commit logs are present. For example, \$CASSANDRA/data/cdc_raw_staged/CommitLog-6-1524110205396.log).
userName	Default: None Mandatory: No Legal Values: Valid SFTP auth username Description: SFTP User name to connect to the server.
password	Default: None Mandatory: No Legal Values: Valid SFTP auth password Description: SFTP password to connect to the server.

Table 10-2 (Cont.) Argument `cassCommitLogPurgerConfFile`

Parameters	Description
<code>port</code>	Default: 22 Mandatory: No Legal Values: Valid SFTP auth port Description: SFTP port number
<code>privateKey</code>	Default: None Mandatory: No Legal Values: valid path to the <code>privateKey</code> file Description: The private key is used to perform the authentication, allowing you to log in without having to specify a password. Providing the <code>privateKey</code> file path allows the purger program to access the nodes with out password.
<code>passPhase</code>	Default: None Mandatory: No Legal Values: valid password for <code>privateKey</code> Description: The private key is typically password protected. If it is provided, then the <code>passPhase</code> with <code>privateKey</code> and <code>passPhase</code> are required to be passed with the password that helps the purger program to successfully access the nodes.

- [Sample config.properties for Local File System](#)
- [Sample config.properties for Remote File System](#)

10.1.4.12.1.3.1 Sample config.properties for Local File System

```
fileSystemType=local
chkDir=apache-cassandra-3.11.2/data/chkdir/
cdcStagingDir=apache-cassandra-3.11.2/data/$nodeAddress/commitlog/
```

10.1.4.12.1.3.2 Sample config.properties for Remote File System

```
fileSystemType=remote
chkDir=apache-cassandra-3.11.2/data/chkdir/
cdcStagingDir=apache-cassandra-3.11.2/data/$nodeAddress/commitlog/
username=username
password=@@@@@
port=22
```

10.1.4.12.1.4 Argument `purgeInterval`

Setting the optional argument `purgeInterval` helps in configuring the process to run as a daemon.

This argument is an integer value representing the time period of clean-up to happen. For example, if `purgeInterval` is set to 1, then the process runs every day on the time the process started.

10.1.4.12.1.5 Argument cassUnProcessedFilesPurgeInterval

Setting the optional argument `cassUnProcessedFilesPurgeInterval` helps in purging historical commit logs for all the nodes that do not have a last processed file.

If `cassUnProcessedFilesPurgeInterval` is not set, then the default value is configured to 2 days; the files older than 2 days or as per the configured value days, and the commit log files are purged. The `CassandraCommitLogPurger` Utility can't purge files that are older than a day. It should be either the default 2 days or more than that.

The following is an example of checkpoint file:

```
{
  "start_timestamp": -1,
  "sequence_id": 34010434,
  "updated_datetime": "2018-04-19 23:24:57.164-0700",
  "nodes": [
    { "address": "10.247.136.146", "offset": 0, "id": 0 }

    /
    { "address": "10.247.136.142", "file": "CommitLog-6-1524110205398.log", "offset":
    33554405, "id": 1524110205398 }

    /
    { "address": "10.248.10.24", "file": "CommitLog-6-1524110205399.log", "offset":
    33554406, "id": 1524110205399 }

    /
    { "address": "10.248.10.25", "offset": 0, "id": 0 }

    /
    { "address": "10.248.10.26", "offset": 0, "id": 0 }

  ]
}
```

In this example, the Cassandra nodes addresses 10.248.10.25 and 10.248.10.26 do not have a last processed file. The commit log files in those nodes will be purged as per the configured days of `cassUnProcessedFilesPurgeInterval` argument value.

Note

The last processing file may not be available due to the following reasons:

- A new node was added into the cluster and no commit log files were processed through Cassandra extract yet.
- All the commit log files processed from this node does not contain operation data as per the table wildcard match.
- All the commit log files processed from this node contain operation records that were not written to the trail file due to de-duplication.

10.1.4.13 Multiple Extract Support

Multiple Extract groups in a single Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) installation can be configured to connect to the same Cassandra cluster.

To run multiple Extract groups:

1. One (and only one) Extract group can be configured to move the commit log files in the `cdc_raw` directory on the Cassandra nodes to a staging directory. The `movecommitlogstostagingdir` parameter is enabled by default and no additional configuration is required for this Extract group.
2. All the other Extract groups should be configured with the `nomovecommitlogstostagingdir` parameter in the Extract parameter (`.prm`) file.

10.1.4.14 CDC Configuration Reference

The following properties are used with Cassandra change data capture.

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
DBOPTIONS ENABLE_CPP_DRIVE R_TRACE true	Optio nal	Extract paramet er (.prm) file.	false	Use only during initial load process. When set to true, the Cassandra driver logs all the API calls to a <code>driver.log</code> file. This file is created in the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) installation directory. This is useful for debugging.
DBOPTIONS FETCHBATCHSIZE <i>number</i>	Optio nal	Extract paramet er (.prm) file.	1000 Minimum is 1 Maximu m is 100000	Use only during initial load process. Specifies the number of rows of data the driver attempts to fetch on each request submitted to the database server. The parameter value should be lower than the database configuration parameter, <code>tombstone_warn_threshold</code> , in the database configuration file, <code>cassandra.yaml</code> . Otherwise the initial load process might fail. Oracle recommends that you set this parameter value to 5000 for initial load Extract optimum performance.
TRANLOGOPTIONS CDCLOGDIRTEMPLAT E <i>path</i>	Requ ired	Extract paramet er (.prm) file.	None	The CDC commit log directory path template. The template can optionally have the <code>\$nodeAddress</code> meta field that is resolved to the respective node address.

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
TRANLOGOPTIONS SFTP <i>options</i>	Optio nal	Extract paramet er (.prm) file.	None	The secure file transfer protocol (SFTP) connection details to pull and transfer the commit log files. You can use one or more of these options: USER <i>user</i> The SFTP user name. PASSWORD <i>password</i> The SFTP password. KNOWNHOSTSFILE <i>file</i> The location of the Secure Shell (SSH)known hosts file. LANDINGDIR <i>dir</i> The SFTP landing directory for the commit log files on the local machine. PRIVATEKEY <i>file</i> The SSH private key file. PASSPHRASE <i>password</i> The SSH private key pass phrase. PORTNUMBER <i>portnumber</i> The SSH port number.

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
CLUSTERCONTACTPOINTS <i>nodes</i>	Optio nal	GLOBAL file	127.0.0.1	A comma separated list of nodes to be used for a connection to the Cassandra cluster. You should provide at least one node address. The parameter options are:

PORT *<port number>*

No default

Optional

The port to use when connecting to the database.

Starting from Oracle GoldenGate Distributed App

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
				p l i c a t i o n s a n d A n a l y t i c s (G G f o r D A A) 2 3 a i , t h i s p a r a m e t e r w i

Properties	Required/Optional	Location	Default	Explanation
Illustration of a table with 5 columns: Properties, Required/Optional, Location, Default, and Explanation.				
TRANLOGOPTIONS CDCREADERSDKVERSION <i>version</i>	Optional	Extract parameter (.prm) file.	3.11	The SDK Version for the CDC reader capture API.
ABENDONMISSEDRECORD NOABENDONMISSEDR ECORD	Optional	Extract parameter (.prm) file.	true	When set to true and the possibility of a missing record is found, the process stops with the diagnostic information. This is generally detected when a node goes down and the CDC reader doesn't find a replica node with a matching last record from the dead node. You can set this parameter to false to continue processing. A warning message is logged about the scenario.
TRANLOGOPTIONS CLEANUPCDCCOMMIT LOGS	Optional	Extract parameter (.prm) file.	false	Purge CDC commit log files post extract processing. When the value is set to false, the CDC commit log files are moved to the staging directory for the commit log files.
JVMOPTIONS [CLASSPATH <classpath> BOOTOPTIONS <options>]	Mandatory	Extract parameter (.prm) file.	None	- CLASSPATH: The classpath for the Java Virtual Machine. You can include an asterisk (*) wildcard to match all JAR files in any directory. Multiple paths should be delimited with a colon (:) character. - BOOTOPTIONS: The boot options for the Java Virtual Machine. Multiple options are delimited by a space character.

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
JVMB00TOPTIONS <i>jvm_options</i>	Optio nal	GLOBA LS file	None	The boot options for the Java Virtual Machine. Multiple options are delimited by a space character.

 Note
Starting from Oracle GoldenGate for Distributed Applications

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
				p l i c a t i o n s a n d A n a l y t i c s (G G f o r D A A) 2 3 a i , t h i s p a r a m e t e r w i

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
				l l b e d e p r e c a t e d .

Properties	Required/Optional	Location	Default	Explanation
JVMCLASSPATH <i>classpath</i>	Required	Global file	None	The classpath for the Java Virtual Machine. You can include an asterisk (*) wildcard to match all JAR files in any directory. Multiple paths should be delimited with a colon (:) character.

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
				p l i c a t i o n s a n d A n a l y t i c s (G G f o r D A A) 2 3 a i , t h i s p a r a m e t e r w i

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
				l l b e d e p r e c a t e d .

Properties		Req uire d/ Opti onal	Locatio n	Default	Explanation
OGGSOURCE	<i>source</i>	Requ ired	GLOBA LS file	None	The source database for CDC capture or database queries. The valid value is CASSANDRA.

 Note

Starting from Oracle GoldenGate for Distributed Applications

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
				p l i c a t i o n s a n d A n a l y t i c s (G G f o r D A A) 2 3 a i , t h i s p a r a m e t e r w i

l
l
b
e
d
e
p
r
e
c
a
t
e
d
.

Properties	Required/ Optional	Location	Default	Explanation
MOVECOMMITLOGSTOSTAGINGDIR NOMOVECOMMITLOGSTOSTAGINGDIR	Optional	Extract parameter (.prm) file.	true	Enabled by default and this instructs the Extract group to move the commit log files in the cdc_raw directory on the Cassandra nodes to a staging directory for the commit log files. Only one Extract group can have movecommitlogstostagingdir enabled, and all the other Extract groups disable this by specifying nomovecommitlogstostagingdir.
SSL	Optional	GLOBAL or Extract parameter (.prm) file.	false	Use for basic SSL support during connection. Additional JSSE configuration through Java System properties is expected when enabling this.
CPPDRIVEROPTIONS SSL PEMPUBLICKEYFILE <i>cassandra.pem</i>	Optional	GLOBAL or Extract parameter (.prm) file. String that indicates the absolute path with fully qualified name. This file is must for the SSL connection.	None, unless the PEMPUBLICKEYFILE property is specified, then you must specify a value.	Indicates that it is PEM formatted public key file used to verify the peer's certificate. This property is needed for one-way handshake or basic SSL connection.

Note

The following SSL properties are in CPPDRIVEROPTIONS SSL so this keyword must be added to any other SSL property to work.

Properties	Required/Optional	Location	Default	Explanation
CPPDRIVEROPTIONS SSL ENABLECLIENTAUTH DISABLECLIENTAUTH	Optional	GLOBAL or Extract parameter (.prm) file.	false	Enabled indicates a two-way SSL encryption between client and server. It is required to authenticate both the client and the server through PEM formatted certificates. This property also needs the pemclientpublickeyfile and pemclientprivatekeyfile properties to be set. The pemclientprivatekeypasswd property must be configured if the client private key is password protected. Setting this property to false disables client authentication for two-way handshake.
CPPDRIVEROPTIONS SSL PEMCLIENTPUBLICKEYFILE <i>public.pem</i>	Optional	GLOBAL or Extract parameter (.prm) file. String that indicates the absolute path with fully qualified name. This file is must for the SSL connection.	None, unless the PEMCLIENTPUBLICKEYFILE property is specified, then you must specify a value.	Use for a PEM formatted public key file name used to verify the client's certificate. This is must if you are using CPPDRIVEROPTIONS SSL ENABLECLIENTAUTH or for two-way handshake.

Properties	Required/ Optional	Location	Default	Explanation
CPPDRIVEROPTIONS SSL PEMCLIENTPRIVATE KEYFILE <i>public.pem</i>	Optional	GLOBAL or Extract parameter (.prm) file. String that indicates the absolute path with fully qualified name. This file is must for the SSL connection.	None, unless the PEMCLIENTPRIVATE TEKEYFILE property is specified, then you must specify a value.	Use for a PEM formatted private key file name used to verify the client's certificate. This is must if you are using CPPDRIVEROPTIONS SSL ENABLECLIENTAUTH or for two-way handshake.
CPPDRIVEROPTIONS SSL PEMCLIENTPRIVATE KEYPASSWD <i>privateKeyPasswd</i>	Optional	GLOBAL or Extract parameter (.prm) file. A string	None, unless the PEMCLIENTPRIVATE TEKEYPASSWD property is specified, then you must specify a value.	Sets the password for the PEM formatted private key file used to verify the client's certificate. This is must if the private key file is protected with the password.
CPPDRIVEROPTIONS SSL PEERCERTVERIFICATIONFLAG <i>value</i>	Optional	GLOBAL or Extract parameter (.prm) file. An integer	0	Sets the verification required on the peer's certificate. The range is 0–4: 0–Disable certificate identity verification. 1–Verify the peer certificate 2–Verify the peer identity 3– Not used so it is similar to disable certificate identity verification. 4 –Verify the peer identity by its domain name

Properties	Required/ Optional	Location	Default	Explanation
CPPDRIVEROPTIONS SSL ENABLEREVERSEDNS	Optional	GLOBAL or Extract parameter (.prm) file.	false	Enables retrieving host name for IP addresses using reverse IP lookup.
TRANLOGOPTIONS TRACKSCHEMACHANGES	Optional	Extract parameter (.prm) file.	By default, the property is disabled.	This will enable extract to capture table level DDL changes from the source at runtime. Enable this to ensure that the table metadata within the trail stays in sync with the source without any downtime. When TRACKSCHEMACHANGES is disabled, the capture process will ABEND if a DDL change is detected at the source table.

10.1.4.15 Troubleshooting

No data captured by the Cassandra Extract process.

- The Cassandra database has not flushed the data from the active commit log files to the CDC commit log files. The flush is dependent on the load of the Cassandra cluster.
- The Cassandra Extract captures data from the CDC commit log files only.
- Check the CDC property of the source table. The CDC property of the source table should be set to true.
- Data is not captured if the TRANLOGOPTIONS CDCREADERSDKVERSION 3.9 parameter is in use and the JVMCLASSPATH is configured to point to Cassandra 3.10 or 3.11 JAR files.

Error: OGG-01115 Function getInstance not implemented.

- The following line is missing from the GLOBALS file.
OGGSOURCE CASSANDRA

Error: Unable to connect to Cassandra cluster, Exception: com.datastax.driver.core.exceptions.NoHostAvailableException

This indicates that the connection to the Cassandra cluster was unsuccessful.

Check the following parameters:

CLUSTERCONTACTPOINTS

Error: Exception in thread "main" java.lang.NoClassDefFoundError: oracle/goldengate/capture/cassandra/CassandraCDCProcessManager

Check the JVMOPTIONS CLASSPATH parameter in the GLOBALS file.

Error: oracle.goldengate.util.Util - Unable to invoke method while constructing object. Unable to create object of class "oracle.goldengate.capture.cassandrapture311.SchemaLoader3DOT11" Caused by:

**java.lang.NoSuchMethodError:
org.apache.cassandra.config.DatabaseDescriptor.clientInitialization()V**

There is a mismatch in the Cassandra SDK version configuration. The `TRANLOGOPTIONS CDCREADERSDKVERSION 3.11` parameter is in use and the `JVMCLASSPATH` may have the Cassandra 3.9 JAR file path.

Error: OGG-25171 Trail file '/path/to/trail/gg' is remote. Only local trail allowed for this extract.

A Cassandra Extract should only be configured to write to local trail files. When adding trail files for Cassandra Extract, use the `EXTTRAIL` option. For example:

```
ADD EXTTRAIL ./dirdat/z1, EXTRACT cass
```

Errors: OGG-868 error message or OGG-4510 error message

The cause could be any of the following:

- Unknown user or invalid password
- Unknown node address
- Insufficient memory

Another cause could be that the connection to the Cassandra database is broken. The *error message* indicates the database error that has occurred.

Error: OGG-251712 Keyspace keyspacename does not exist in the database.

The issue could be due to these conditions:

- During the Extract initial load process, you may have deleted the `KEYSPACE keyspacename` from the Cassandra database.
- The `KEYSPACE keyspacename` does not exist in the Cassandra database.

Error: OGG-25175 Unexpected error while fetching row.

This can occur if the connection to the Cassandra database is broken during initial load process.

Error: “Server-side warning: Read 915936 live rows and 12823104 tombstone cells for query `SELECT * FROM keyspace.table`(see `tombstone_warn_threshold`)”.

When the value of the initial load `DBOPTIONS FETCHBATCHSIZE` parameter is greater than the Cassandra database configuration parameter, `tombstone_warn_threshold`, this is likely to occur.

Increase the value of `tombstone_warn_threshold` or reduce the `DBOPTIONS FETCHBATCHSIZE` value to get around this issue.

Duplicate records in the Cassandra Extract trail.

Internal tests on a multi-node Cassandra cluster have revealed that there is a possibility of duplicate records in the Cassandra CDC commit log files. The duplication in the Cassandra commit log files is more common when there is heavy write parallelism, write errors on nodes, and multiple retry attempts on the Cassandra nodes. In these cases, it is expected that Cassandra trail file will have duplicate records.

JSchException or SftpException in the Extract Report File

Verify that the SFTP credentials (user, password, and privatekey) are correct. Check that the SFTP user has read and write permissions for the cdc_raw directory on each of the nodes in the Cassandra cluster.

ERROR o.g.c.c.CassandraCDCProcessManager - Exception during creation of CDC staging directory [{}]]java.nio.file.AccessDeniedException

The Extract process does not have permission to create CDC commit log staging directory. For example, if the cdc_raw commit log directory is /path/to/cassandra/home/data/cdc_raw, then the staging directory would be /path/to/cassandra/home/data/cdc_raw/./cdc_raw_staged.

Extract report file shows a lot of DEBUG log statements

On production system, you do not need to enable debug logging. To use INFO level logging, make sure that the Extract parameter file include this

```
JVMBOOTOPTIONS -Dlogback.configurationFile=AdapterExamples/big-data/cassandrapture/logback.xml
```

To enable SSL in Oracle Golden Gate Cassandra Extract you have to enable SSL in the GLOBALS file or in the Extract Parameter file.

If SSL Keyword is missing, then Extract assumes that you wanted to connect without SSL. So if the Cassandra.yaml file has an SSL configuration entry, then the connection fails.

SSL is enabled and it is one-way handshake

You must specify the CPPDRIVEROPTIONS SSL PEMPUBLICKEYFILE /scratch/testcassandra/testssl/ssl/cassandra.pem property.

If this property is missing, then Extract generates this error:.

```
2018-06-09 01:55:37 ERROR OGG-25180 The PEM formatted public key file used to verify the peer's certificate is missing.
```

If SSL is enabled, then it is must to set PEMPUBLICKEYFILE in your Oracle GoldenGate GLOBALS file or in Extract parameter file

SSL is enabled and it is two-way handshake

You must specify these properties for SSL two-way handshake:

```
CPPDRIVEROPTIONS SSL ENABLECLIENTAUTH
CPPDRIVEROPTIONS SSL PEMCLIENTPUBLICKEYFILE /scratch/testcassandra/testssl/ssl/datastax-cppdriver.pem
CPPDRIVEROPTIONS SSL PEMCLIENTPRIVATEKEYFILE /scratch/testcassandra/testssl/ssl/datastax-cppdriver-private.pem
CPPDRIVEROPTIONS SSL PEMCLIENTPRIVATEKEYPASSWD cassandra
```

Additionally, consider the following:

- If ENABLECLIENTAUTH is missing then Extract assumes that it is one-way handshake so it ignores PEMCLIENTPRIVATEKEYFILE and PEMCLIENTPRIVATEKEYFILE. The following error occurs because the cassandra.yaml file should have require_client_auth set to true.

```
2018-06-09 02:00:35 ERROR OGG-00868 No hosts available for the control connection.
```

- If ENABLECLIENTAUTH is used and PEMCLIENTPRIVATEKEYFILE is missing, then this error occurs:

2018-06-09 02:04:46 ERROR OGG-25178 The PEM formatted private key file used to verify the client's certificate is missing. For two way handshake or if ENABLECLIENTAUTH is set, then it is mandatory to set PEMCLIENTPRIVATEKEYFILE in your Oracle GoldenGate GLOBALS file or in Extract parameter file.
- If ENABLECLIENTAUTH is use and PEMCLIENTPUBLICKEYFILE is missing, then this error occurs:

2018-06-09 02:06:20 ERROR OGG-25179 The PEM formatted public key file used to verify the client's certificate is missing. For two way handshake or if ENABLECLIENTAUTH is set, then it is mandatory to set PEMCLIENTPUBLICKEYFILE in your Oracle GoldenGate GLOBALS file or in Extract parameter file.
- If the password is set while generating the client private key file then you must add PEMCLIENTPRIVATEKEYPASSWD to avoid this error:

2018-06-09 02:09:48 ERROR OGG-25177 The SSL certificate: /scratch/jitiwari/testcassandra/testssl/ssl/datastax-cppdriver-private.pem can not be loaded. Unable to load private key.
- If any of the PEM file is missing from the specified absolute path, then this error occurs:

2018-06-09 02:12:39 ERROR OGG-25176 Can not open the SSL certificate: /scratch/jitiwari/testcassandra/testssl/ssl/cassandra.pem.

com.jcraft.jsch.JSchException: UnknownHostKey

If the extract process ABENDs with this issue, then it is likely that some or all the Cassandra node addresses are missing in the SSH known-hosts file. For more information, see [Setup SSH Connection to the Cassandra Nodes](#).

General SSL Errors

Consider these general errors:

- The SSL connection may fail if you have enabled all SSL required parameters in Extract or GLOBALS file and the SSL is not configured in the `cassandra.yaml` file.
- The absolute path or the qualified name of the PEM file may not correct. There could be access issue on the PEM file stored location.
- The password added during generating the client private key file may not be correct or you may not have enabled it in the Extract parameter or GLOBALS file.

10.1.4.16 Cassandra Capture Client Dependencies

What are the dependencies for the Cassandra Capture (Extract) to connect to Apache Cassandra databases?

The following third party libraries are needed to run Cassandra Change Data Capture.

Capturing from Apache Cassandra 3.x versions:

- `cassandra-driver-core` (com.datastax.cassandra) version 3.3.1
- `cassandra-all` (org.apache.cassandra) version 3.11.0
- `gson` (com.google.code.gson) version 2.8.0
- `jsch` (com.jcraft) version 0.1.54

Capturing from Apache Cassandra 4.x versions:

- java-driver-core (com.datastax.oss) version 4.14.1
- cassandra-all (org.apache.cassandra) version 4.0.5
- gson (com.google.code.gson) version 2.8.0
- jsch (com.jcraft) version 0.1.54

You can use the Dependency Downloader scripts to download the Datastax Java Driver and its associated dependencies. For more information, see [Dependency Downloader](#).

10.1.5 Apache Kafka

The Oracle GoldenGate capture (Extract) for Kafka is used to read messages from a Kafka topic or topics and convert data into logical change records written to GoldenGate trail files. This section explains how to use Oracle GoldenGate capture for Kafka.

- [Overview](#)
- [Prerequisites](#)
- [General Terms and Functionality of Kafka Capture](#)
- [Generic Mutation Builder](#)
- [Kafka Connect Mutation Builder](#)
- [Example Configuration Files](#)

10.1.5.1 Overview

Kafka has gained market traction in recent years and become a leader in the enterprise messaging space. Kafka is a cluster-based messaging system that provides high availability, fail over, data integrity through redundancy, and high performance. Kafka is now the leading application for implementations of the Enterprise Service Bus architecture. Kafka Capture extract process reads messages from Kafka and transforms those messages into logical change records which are written to Oracle GoldenGate trail files. The generated trail files can then be used to propagate the data in the trail file to various RDBMS implementations or other integrations supported by Oracle GoldenGate replicat processes.

10.1.5.2 Prerequisites

- [Set up Credential Store Entry to Detect Source Type](#)

10.1.5.2.1 Set up Credential Store Entry to Detect Source Type

The database type for capture is based on the prefix in the database credential userid. The generic format for userid is as follows: `<dbtype>://<db-user>@<comma separated list of server addresses>:<port>`

The userid value for Kafka capture should be any value with the prefix `kafka://`. You can set up Credential Store Entry in **Administration Service/ DB Connections**.

Example

```
alter credentialstore add user kafka:// password somepass alias kafka
```

Note

You can specify a dummy Password for Kafka while setting up the credentials.

10.1.5.3 General Terms and Functionality of Kafka Capture

- [Kafka Streams](#)
- [Kafka Message Order](#)
- [Kafka Message Timestamps](#)
- [Kafka Message Coordinates](#)
- [Start Extract Modes](#)
- [General Configuration Overview](#)
- [OGGSOURCE parameter](#)
- [The Extract Parameter File](#)
- [Kafka Consumer Properties File](#)

10.1.5.3.1 Kafka Streams

As a Kafka consumer, you can read from one or more topics. Additionally, each topic can be divided into one or more partitions. Each discrete topic/partition combination is a Kafka stream. This topic discusses Kafka streams extensively and it is important to clearly define the term here.

The following is an example of five Kafka streams:

- Topic: TEST1 Partition: 0
- Topic: TEST1 Partition: 1
- Topic: TEST2 Partition: 0
- Topic: TEST2 Partition: 1
- Topic: TEST2 Partition: 2

10.1.5.3.2 Kafka Message Order

Messages received from the KafkaConsumer for an individual stream should be in the order as stored in the Kafka commit log. However, Kafka streams move independently from one another and the order in which messages are received from different streams is nondeterministic.

For example, Kafka Capture is consuming messages from two streams:

- Stream 1: Topic TEST1, partition 0
- Stream 2: Topic TEST1, partition 1

Stream 1 in Topic|partition|offset|timestamp format total of 5 messages.

```
TEST1|0|0|1588888086210
TEST1|0|1|1588888086220
TEST1|0|2|1588888086230
TEST1|0|3|1588888086240
TEST1|0|4|1588888086250
```

Stream 2 to Topic|partition|offset|timestamp format total of 5 messages.

```
TEST1|1|0|1588888086215
TEST1|1|1|1588888086225
TEST1|1|2|1588888086235
```

```
TEST1|1|3|1588888086245
TEST1|1|4|1588888086255
```

The Kafka Consumer could deliver the messages in the following order on run 1.

```
TEST1|1|0|1588888086215
TEST1|1|1|1588888086225
TEST1|0|0|1588888086210
TEST1|0|1|1588888086220
TEST1|0|2|1588888086230
TEST1|0|3|1588888086240
TEST1|0|4|1588888086250
TEST1|1|2|1588888086235
TEST1|1|3|1588888086245
TEST1|1|4|1588888086255
```

On a secondary run messages could be delivered in the following order.

```
TEST1|0|0|1588888086210
TEST1|0|1|1588888086220
TEST1|1|0|1588888086215
TEST1|1|1|1588888086225
TEST1|0|2|1588888086230
TEST1|0|3|1588888086240
TEST1|0|4|1588888086250
TEST1|1|2|1588888086235
TEST1|1|3|1588888086245
TEST1|1|4|1588888086255
```

Note

In the two runs that the messages belonging to the same Kafka stream are delivered in order as they occur in that stream. However, messages from different streams are interlaced in a nondeterministic manner.

10.1.5.3.3 Kafka Message Timestamps

Each Kafka message has a timestamp associated with it. The timestamp on the Kafka message maps to the operation timestamp for the record in the generated trail file. Timestamps on Kafka messages are not guaranteed to be monotonically increasing even in the case where extract is reading from only one stream (single topic and partition). Kafka has no requirement that Kafka message timestamps are monotonically increasing even within a stream. The Kafka Producer provides an API whereby the message timestamp can be explicitly set on messages. This means a Kafka Producer can set the Kafka message timestamp to any value.

When reading from multiple topics and/or a topic with multiple partitions it is almost certain that trail files generated by Kafka capture will not have operation timestamps that are monotonically increasing. Kafka streams move independently from one another and there is no guarantee of delivery order for messages received from different streams. Messages from different streams can interlace in any random order when the Kafka Consumer is reading them from a Kafka cluster.

10.1.5.3.4 Kafka Message Coordinates

Kafka Capture performs message gap checking to ensure message consistency withing the context of a message stream. For every Kafka stream from which Kafka capture is consuming messages, there should be no gap in the Kafka message offset sequence.

If a gap is found in the message offset sequence, then the Kafka capture logs an error and the Kafka Capture extract process will abend.

Message gap checking can be disabled by setting the following in the parameter file.

SETENV (PERFORMMESSAGEGAPCHECK = "false").

10.1.5.3.5 Start Extract Modes

Extract can be configured to start replication from two distinct points. You can select Extract starting positions from the UI in the Extract Options step, under the Begin section. You can either click **Now** or define a **Custom Time**.

- [Start Earliest](#)
- [Start Timestamp](#)

10.1.5.3.5.1 Start Earliest

Start Kafka Capture from the oldest available message in Kafka.

```
ggsci> ADD EXTRACT kafka, TRANLOG
ggsci> ADD EXTRAIL dirdat/kc, extract kafka
ggsci> START EXTRACT kafka
```

10.1.5.3.5.2 Start Timestamp

Start Kafka Capture from the oldest available message in Kafka.

```
ggsci> ADD EXTRACT kafka, TRANLOG BEGIN 2019-03-27 23:05:05.123456
ggsci> ADD EXTRAIL dirdat/kc, extract kafka
ggsci> START EXTRACT kafka
```

Or alternatively, start now as now is a point in time.

```
ggsci> ADD EXTRACT kafka, TRANLOG BEGIN NOW
ggsci> ADD EXTRAIL dirdat/kc, extract kafka
ggsci> START EXTRACT kafka
```

Note

Note on starting from a point in time. Kafka Capture will start from the first available record in the stream which fits the criteria (time equal to or greater than the configured time). Replicat will continue from that first message regardless of the timestamps of subsequent messages. As previously discussed, there is no guarantee or requirement that Kafka message timestamps are monotonically increasing.

Alter Extract

Alter Timestamp

```
ggsci> STOP EXTRACT kafka
ggsci> ALTER EXTRACT kafka BEGIN {Timestamp}
ggsci> START EXTRACT kafka
```

Alter Now

```
ggsci> STOP EXTRACT kafka
ggsci> ALTER EXTRACT kafka BEGIN NOW
ggsci> START EXTRACT kafka
```

10.1.5.3.6 General Configuration Overview

10.1.5.3.7 OGGSOURCE parameter

To enable Kafka extract replication, the GLOBALS parameter file must be configured as follows:

```
OGGSOURCE KAFKA
JVMCLASSPATH ggjava/ggjava.jar:/kafka/client/path/*:dirprm
JVMBOOTOPTIONS -Xmx512m -Dlog4j.configurationFile=log4j-default.properties -
Dgg.log.level=INFO
```

OGGSOURCE KAFKA: The first line indicates that the source of replication is Kafka.

JVMCLASSPATH ggjava/ggjava.jar:/kafka/client/path/*:dirprm: The second line sets the Java JVM classpath. The Java classpath provides the pathing to load all the required Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) and Kafka client libraries. The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) library should be first in the list (ggjava.jar). The Kafka client libraries, the Kafka Connect framework, and the Kafka Connect converters are not included with the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) installation. These libraries must be obtained independently. Oracle recommends you to use the same version of the Kafka client as the version of the Kafka broker to which you are connecting. The Dependency Downloading tool can be used to download the dependency libraries. Alternately, the pathing can be set to a Kafka installation. For more information about Dependency Downloader, see [Dependency Downloader](#).

JVMBOOTOPTIONS -Xmx512m -Dlog4j.configurationFile=log4j-default.properties -Dgg.log.level=INFO: The third line is the JVM boot options. Use this to configure the maximum Java heap size (-Xmx512m) and the log4j logging parameters to generate the .log file (-Dlog4j.configurationFile=log4j-default.properties -Dgg.log.level=INFO)

Note

Starting from Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) release 23c, this parameter will be deprecated.

10.1.5.3.8 The Extract Parameter File

The extract process configured is configured via a .prm file. The format for the naming of the parameter file is <extract name>.prm. For example, the extract parameter file for the extract process kc would be kc.prm.

```
EXTRACT KC
-- alter credentialstore add user kafka:// password <somepass> alias kafka
SOURCEDB USERIDALIAS kafka
JVMOPTIONS CLASSPATH ggjava/ggjava.jar:/kafka/client/path/*
JVMOPTIONS BOOTOPTIONS -Xmx512m -Dlog4j.configurationFile=log4j-default.properties -
Dgg.log.level=INFO
TRANLOGOPTIONS GETMETADATAFROMVAM
TRANLOGOPTIONS KAFKACONSUMERPROPERTIES kafka_consumer.properties
EXTTRAIL dirdat/kc
TABLE QASOURCE.TOPIC1;
```

EXTRACT KC: The first line sets the name of the extract process.

TRANLOGOPTIONS KAFKACONSUMERPROPERTIES kafka_consumer.properties: This line sets the name and location of the Kafka Consumer properties file. The Kafka Consumer properties is a file containing the Kafka specific configuration which configures connectivity and security to the Kafka cluster. Documentation on the Kafka Consumer properties can be found in: [Kafka Documentation](#).

EXTTRAIL dirdat/kc: The fourth line sets the location and prefix of the trail files to be generated.

TABLE QASOURCE.TOPIC1;: The fifth line is the extract TABLE statement. There can be one or more TABLE statements. The schema name in the example is QASOURCE. The schema name is an OGG artifact and it is required. It can be set to any legal string. The schema name cannot be wildcarded. Each extract process only supports one schema name. The configured table name maps to the Kafka topic name. The table configuration does support wildcards. Legal Kafka topic names can have the following characters.

- a-z (lowercase a to z)
- A-Z (uppercase A to Z)
- 0-9 (digits 0 to 9)
- . (period)
- _ (underscore)
- - (hyphen)

If the topic name contains a period, underscore, or hyphen, please include the table name in quotes in the configuration. Topic names are case sensitive so the topic MYTOPIC1 and MyTopic1 are different Kafka topics.

Examples of legal extract table statements:

```
TABLE TESTSCHEMA.TEST*;
TABLE TESTSCHEMA.MyTopic1;
TABLE TESTSCHEMA."My.Topic1";
```

Examples of illegal configuration - multiple schema names are used:

```
TABLE QASOURCE.TEST*;
TABLE TESTSCHEMA.MYTOPIC1;
```

Example of illegal configuration – Table with special characters not quoted.

```
TABLE QASOURE.My.Topic1;
```

Example of illegal configuration – Schema name is a wildcard.

```
TABLE *. *;
```

Optional .prm configuration.

Kafka Capture performs message gap checking to ensure message continuity. To disable message gap checking set:

```
SETENV (PERFORMMESSAGEGAPCHECK = "false")
```

10.1.5.3.9 Kafka Consumer Properties File

The Kafka Consumer properties file contains the properties to configure the Kafka Consumer including how to connect to the Kafka cluster and security parameters.

Example:

```
#Kafka Properties
bootstrap.servers=den02box:9092
group.id=mygroupid
key.deserializer=org.apache.kafka.common.serialization.ByteArrayDeserializer
value.deserializer=org.apache.kafka.common.serialization.ByteArrayDeserializer
```

- [Encrypt Kafka Producer Properties](#)

10.1.5.3.9.1 Encrypt Kafka Producer Properties

The sensitive properties within the Kafka Producer Configuration File can be encrypted using the Oracle GoldenGate Credential Store.

For more information about how to use Credential Store, see [Using Identities in Oracle GoldenGate Credential Store](#).

For example, the following kafka property:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule
required
username="alice" password="alice";
```

can be replaced with:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule
required
username=ORACLEWALLETUSERNAME[alias domain_name]
password=ORACLEWALLETPASSWORD[alias
domain_name];
```

10.1.5.4 Generic Mutation Builder

The default mode is to use the Generic Mutation Builder to transform Kafka messages into trail file operations. Kafka messages are comprised of data in any format. Kafka messages can be delimited text, JSON, Avro, XML, text, etc. This makes the mapping of data from a Kafka message into a logical change record challenging. However, Kafka message keys and payload values are at their fundamental form just byte arrays. The Generic Kafka Replication simply propagates the Kafka message key and Kafka message value as byte arrays. Generic Kafka Replication transforms the data into operations of three fields. The three fields are as follows:

- **id:** This is the primary key for the table. It is typed as a string. The value is the coordinates of the message in Kafka in the following format: topic name:partition number:offset. For example, the value for topic TEST, partition 1, and offset 245 would be TEST:1:245.
- **key:** This is the message key field from the source Kafka message. The field is typed as binary. The value of the field is the key from the source Kafka message propagated as bytes.
- **payload:** This is the message payload or value from the source Kafka message. The field is typed as binary. The value of the field is the payload from the source Kafka message propagated as bytes.

Features of the Generic Mutation Builder

- All records are propagated as insert operations.
- Each Kafka message creates an operation in its own transaction.

Logdump 2666 >n

```

Hdr-Ind      :      E (x45)      Partition :      . (x00)
UndoFlag     :      . (x00)      BeforeAfter:      A (x41)
RecLength    :    196 (x00c4)    IO Time    : 2021/07/22 14:57:25.085.436
IOType       :    170 (xaa)      OrigNode   :      2 (x02)
TransInd     :      . (x03)      FormatType :      R (x52)
SyskeyLen    :      0 (x00)      Incomplete:      . (x00)
DDR/TDR index: (001, 001)      AuditPos    :      0
Continued    :      N (x00)      RecCount   :      1 (x01)

```

2021/07/22 14:57:25.085.436 Metadata Len 196 RBA 1335

Table Name: QASOURCE.TOPIC1

*

```

1)Name          2)Data Type          3)External Length  4)Fetch Offset
5)Scale          6)Level
7)Null           8)Bump if Odd        9)Internal Length 10)Binary Length  11)Table
Length 12)Most Sig DT
13)Least Sig DT 14)High Precision 15)Low Precision  16)Elementary Item
17)Occurs        18)Key Column
19)Sub DataType 20)Native DataType 21)Character Set  22)Character Length 23)LOB
Type            24)Partial Type
25)Remarks

```

*

TDR version: 11

Definition for table QASOURCE.TOPIC1

Record Length: 20016

Columns: 3

```

id      64  8000      0 0 0 0 0  8000  8000      0 0 0 0 0 1  0 1  0
12      -1    0 0 0
key     64 16000      8005 0 0 1 0  8000  8000      0 0 0 0 0 1  0 0  4
-3      -1    0 0 0
payload 64  8000  16010 0 0 1 0  4000  4000      0 0 0 0 0 1  0 0  4
-4      -1    0 1 0

```

End of definition

10.1.5.5 Kafka Connect Mutation Builder

The Kafka Connect Mutation Builder parses Kafka Connect messages into logical change records and that are then written to Oracle GoldenGate trail files.

- [Functionality and Limitations of the Kafka Connect Mutation Builder](#)
- [Primary Key](#)
- [Kafka Message Key](#)
- [Kafka Connect Supported Types](#)
- [How to Enable the Kafka Connect Mutation Builder](#)

10.1.5.5.1 Functionality and Limitations of the Kafka Connect Mutation Builder

- All records are propagated as insert operations.
- Each Kafka message creates an operation in its own transaction.
- The Kafka message key must be a Kafka Connect primitive type or logical type.
- The Kafka message value must be either a primitive type/logical type or a record containing only primitive types, logical types, and container types. A record cannot contain another record as nested records are not currently supported.
- Kafka Connect array data types are mapped into binary fields. The content of the binary field will be the source array converted into a serialized JSON array.

- Kafka Connect map data types are mapped into binary fields. The contents of the binary field will be the source map converted into a serialized JSON.
- The source Kafka messages must be Kafka Connect messages.
- Kafka Connect Protobuf messages are not currently supported. (The current Kafka Capture functionality only supports primitive or logical types for the Kafka message key. The Kafka Connect Protobuf Converter does not support stand only primitives or logical types.)
- Each source topic must contain messages which conform to the same schema. Interlacing messages in the same Kafka topic which conform to different Kafka Connect schema is not currently supported.
- Schema changes are not currently supported.

10.1.5.5.2 Primary Key

A primary key field is created in the output as a column named `gg_id`. The value of this field is the concatenated topic name, partition, and offset delimited by the `:` character. For example: `TOPIC1:0:1001`.

10.1.5.5.3 Kafka Message Key

The message key is mapped into a column named `gg_key`.

10.1.5.5.4 Kafka Connect Supported Types

Supported Primitive Types

- String
- 8 bit Integer
- 16 bit Integer
- 32 bit Integer
- 64 bit Integer
- Boolean
- 32 bit Float
- 64 bit Float
- Bytes (binary)

Supported Logical Types

- Decimal
- Timestamp
- Date
- Time

Supported Container Types

- Array – Only arrays of primitive or logical types are supported. Data is mapped as a binary field the value of which is a JSON array document containing the contents of the source array.
- List – Only lists of primitive or logical types are supported. Data is mapped as a binary field the value of which is a JSON document containing the contents of the source list.

10.1.5.5.5 How to Enable the Kafka Connect Mutation Builder

The Kafka Connect Mutation Builder is enabled by configuration of the Kafka Connect key and value converters in the Kafka Producer properties file.

For the Kafka Connect JSON Converter

```
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
```

For the Kafka Connect Avro Converter

```
key.converter=io.confluent.connect.avro.AvroConverter
value.converter=io.confluent.connect.avro.AvroConverter
key.converter.schema.registry.url=http://localhost:8081
value.converter.schema.registry.url=http://localhost:8081
```

The Kafka Capture functionality reads the Kafka producer properties file. If the Kafka Connect converters are configured, then the Kafka Connect mutation builder is invoked.

Sample metadata from the trail file using logdump

```
2021/08/03 09:06:05.243.881 Metadata          Len 1951 RBA 1335
Table Name: TEST.KC
*
  1)Name          2)Data Type          3)External Length  4)Fetch Offset
  5)Scale          6)Level
  7)Null           8)Bump if Odd          9)Internal Length 10)Binary Length  11)Table
Length 12)Most Sig DT
13)Least Sig DT 14)High Precision 15)Low Precision  16)Elementary Item
17)Occurs        18)Key Column
19)Sub DataType 20)Native DataType 21)Character Set  22)Character Length 23)LOB
Type            24)Partial Type
25)Remarks
*
TDR version: 11
Definition for table TEST.KC
Record Length: 36422
Columns: 30
gg_id          64  8000          0  0  0  0  0  8000  8000          0  0  0  0  0  1  0
1  0  12      -1    0  0  0
gg_key         64  4000          8005  0  0  1  0  4000  4000          0  0  0  0  0  1  0
0  0  -1      -1    0  1  0
string_required 64  4000        12010  0  0  1  0  4000  4000          0  0  0  0  0  1  0
0  0  -1      -1    0  1  0
string_optional 64  4000        16015  0  0  1  0  4000  4000          0  0  0  0  0  1  0
0  0  -1      -1    0  1  0
byte_required   134  23        20020  0  0  1  0    8    8          8  0  0  0  0  1  0
0  0  4       -1    0  0  0
byte_optional   134  23        20031  0  0  1  0    8    8          8  0  0  0  0  1  0
0  0  4       -1    0  0  0
short_required  134  23        20042  0  0  1  0    8    8          8  0  0  0  0  1  0
0  0  4       -1    0  0  0
short_optional  134  23        20053  0  0  1  0    8    8          8  0  0  0  0  1  0
0  0  4       -1    0  0  0
integer_required 134  23        20064  0  0  1  0    8    8          8  0  0  0  0  1  0
0  0  4       -1    0  0  0
integer_optional 134  23        20075  0  0  1  0    8    8          8  0  0  0  0  1  0
0  0  4       -1    0  0  0
long_required   134  23        20086  0  0  1  0    8    8          8  0  0  0  0  1  0
0  0  -5      -1    0  0  0
long_optional   134  23        20097  0  0  1  0    8    8          8  0  0  0  0  1  0
```

```

0 0 -5 -1 0 0 0
boolean_required 0 2 20108 0 0 1 0 1 1 0 0 0 0 0 1 0
0 4 -2 -1 0 0 0
boolean_optional 0 2 20112 0 0 1 0 1 1 0 0 0 0 0 1 0
0 4 -2 -1 0 0 0
float_required 141 50 20116 0 0 1 0 8 8 8 0 0 0 0 1 0
0 0 6 -1 0 0 0
float_optional 141 50 20127 0 0 1 0 8 8 8 0 0 0 0 1 0
0 0 6 -1 0 0 0
double_required 141 50 20138 0 0 1 0 8 8 8 0 0 0 0 1 0
0 0 8 -1 0 0 0
double_optional 141 50 20149 0 0 1 0 8 8 8 0 0 0 0 1 0
0 0 8 -1 0 0 0
bytes_required 64 8000 20160 0 0 1 0 4000 4000 0 0 0 0 0 1 0
0 4 -4 -1 0 1 0
bytes_optional 64 8000 24165 0 0 1 0 4000 4000 0 0 0 0 0 1 0
0 4 -4 -1 0 1 0
decimal_required 64 50 28170 0 0 1 0 50 50 0 0 0 0 0 1 0
0 0 12 -1 0 0 0
decimal_optional 64 50 28225 0 0 1 0 50 50 0 0 0 0 0 1 0
0 0 12 -1 0 0 0
timestamp_required 192 29 28280 0 0 1 0 29 29 29 0 6 0 0 1 0
0 0 11 -1 0 0 0
timestamp_optional 192 29 28312 0 0 1 0 29 29 29 0 6 0 0 1 0
0 0 11 -1 0 0 0
date_required 192 10 28344 0 0 1 0 10 10 10 0 2 0 0 1 0
0 0 9 -1 0 0 0
date_optional 192 10 28357 0 0 1 0 10 10 10 0 2 0 0 1 0
0 0 9 -1 0 0 0
time_required 192 18 28370 0 0 1 0 18 18 18 3 6 0 0 1 0
0 0 10 -1 0 0 0
time_optional 192 18 28391 0 0 1 0 18 18 18 3 6 0 0 1 0
0 0 10 -1 0 0 0
array_optional 64 8000 28412 0 0 1 0 4000 4000 0 0 0 0 0 1 0
0 4 -4 -1 0 1 0
map_optional 64 8000 32417 0 0 1 0 4000 4000 0 0 0 0 0 1 0
0 4 -4 -1 0 1 0
End of definition

```

10.1.5.6 Example Configuration Files

- [Example kc.prm file](#)
- [Example Kafka Consumer Properties File](#)

10.1.5.6.1 Example kc.prm file

```

EXTRACT KC
OGGSOURCE KAFKA
JVMOPTIONS CLASSPATH ggjava/ggjava.jar:/path/to/kafka/libs/*
TRANLOGOPTIONS GETMETADATAFROMVAM
--Uncomment the following line to disable Kafka message gap checking.
--SETENV (PERFORMMESSAGEGAPCHECK = "false")
TRANLOGOPTIONS KAFKAConsumerProperties kafka_consumer.properties
EXTTRAIL dirdat/kc
TABLE TEST.KC;

```

10.1.5.6.2 Example Kafka Consumer Properties File

```

#Kafka Properties
bootstrap.servers=localhost:9092

```

```
group.id=someuniquevalue
key.deserializer=org.apache.kafka.common.serialization.ByteArrayDeserializer
value.deserializer=org.apache.kafka.common.serialization.ByteArrayDeserializer

#JSON Converter Settings
#Uncomment to use the Kafka Connect Mutation Builder with JSON Kafka Connect Messages
#key.converter=org.apache.kafka.connect.json.JsonConverter
#value.converter=org.apache.kafka.connect.json.JsonConverter

#Avro Converter Settings
#Uncomment to use the Kafka Connect Mutation Builder with Avro Kafka Connect Messages
#key.converter=io.confluent.connect.avro.AvroConverter
#value.converter=io.confluent.connect.avro.AvroConverter
#key.converter.schema.registry.url=http://localhost:8081
#value.converter.schema.registry.url=http://localhost:8081
```

10.1.6 Azure Event Hubs

To capture messages from Azure Event Hubs and parse into logical change records with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA), you can use Kafka Extract. For more information, see [Apache Kafka](#) as source.

10.1.7 Confluent Kafka

To capture Kafka Connect messages from Confluent Kafka and parse into logical change records with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA), you can use Kafka Connect Mutation Builder. For more information, see [Kafka Connect Mutation Builder](#).

10.1.8 DataStax

Datastax Enterprise is a NoSQL database built on Apache Cassandra. For more information, see [Apache Cassandra](#) for configuring change data capture from Datastax Enterprise.

10.1.9 Java Message Service (JMS)

This article explains using the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) to capture Java Message Service (JMS) messages to be written to an Oracle GoldenGate trail.

- [Prerequisites](#)
- [Oracle GoldenGate Java Delivery](#)
- [Configuring Message Capture](#)

10.1.9.1 Prerequisites

- [Set up Credential Store Entry to Detect Source Type](#)

10.1.9.1.1 Set up Credential Store Entry to Detect Source Type

JMS Capture

Similar to Kafka, for the sake of detecting the source type, user can create a credential store entry with the prefix: `jms://`.

Note

You can set up Credential Store Entry in **Administration Service/ DB Connections**.

Example

```
alter credentialstore add user jms:// password <anypassword> alias jms
```

If the extract parameter file does not specify SOURCEDB parameter with USERIDALIAS option, then the source type will be assumed to be JMS, and a warning message will be logged to indicate this.

10.1.9.2 Oracle GoldenGate Java Delivery

Through the Oracle GoldenGate Java API, transactional data captured by Oracle GoldenGate can be delivered to targets other than a relational database, such as a JMS (Java Message Service), files written to disk, streaming data to a Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) application, or integration with a custom application Java API. Oracle GoldenGate Java Delivery can work with either an Extract or Replicat process. Using the Oracle GoldenGate Replicat process is considered the best practice. Oracle GoldenGate Java Delivery requires Java 8 as a dependency.

Oracle GoldenGate for Java provides the ability to execute Java code from the Oracle GoldenGate Replicat process. Using Oracle GoldenGate for Java requires the following conditions to be met:

- A dynamically linked or shared library, implemented in C/C++, integrating an extension module of Oracle GoldenGate Replicat process.
- A set of Java libraries (JARs), which comprise the Oracle GoldenGate Java API. This Java framework communicates with the Replicat through the Java Native Interface (JNI).
- Java 8 must be installed and accessible on the machine hosting the Oracle GoldenGate Java Delivery process or processes. Environmental variables must be correctly set to resolve Java and its associated libraries.

10.1.9.3 Configuring Message Capture

This chapter explains how to configure the VAM Extract to capture JMS messages.

- [Oracle GoldenGate VAM Message Capture](#)
- [Oracle GoldenGate VAM Message Capture](#)
- [Configuring the VAM Extract](#)
- [Connecting and Retrieving the Messages](#)

10.1.9.3.1 Oracle GoldenGate VAM Message Capture

Oracle GoldenGate VAM Message Capture only works with the Oracle GoldenGate Extract process. Oracle GoldenGate message capture connects to JMS messaging to parse messages and send them through a VAM interface to an Oracle GoldenGate Extract process that builds an Oracle GoldenGate trail of message data. This allows JMS messages to be delivered to an Oracle GoldenGate system running for a target database. Java 8 is a required dependency for Oracle GoldenGate VAM Message Capture.

Using Oracle GoldenGate JMS message capture requires the dynamically linked shared VAM library that is attached to the Oracle GoldenGate Extract process.

- [Message Capture Configuration Options](#)
- [Typical Configuration](#)

10.1.9.3.1.1 Message Capture Configuration Options

The options for configuring the three parts of message capture are:

- **Message connectivity:** Values in the property file set connection properties such as the Java classpath for the JMS client, the JMS source destination name, JNDI connection properties, and security information.
- **Parsing:** Values in the property file set parsing rules for fixed width, comma delimited, or XML messages. This includes settings such as the delimiter to be used, values for the beginning and end of transactions and the date format.
- **VAM interface:** Parameters that identify the VAM, dll, or so library and a property file are set for the Oracle GoldenGate core Extract process.

10.1.9.3.1.2 Typical Configuration

In a typical configuration, an Oracle GoldenGate Java Adapter receives JMS Messages. JMS Producers create messages that are received by the Oracle GoldenGate JMS Handler. The messages are then sent using the message capture VAM to an Extract process that writes the data out to an Oracle GoldenGate trail. This trail is sent to a target Oracle GoldenGate instance where it updates the target database.

10.1.9.3.2 Oracle GoldenGate VAM Message Capture

Oracle GoldenGate VAM Message Capture only works with the Oracle GoldenGate Extract process. Oracle GoldenGate message capture connects to JMS messaging to parse messages and send them through a VAM interface to an Oracle GoldenGate Extract process that builds an Oracle GoldenGate trail of message data. This allows JMS messages to be delivered to an Oracle GoldenGate system running for a target database. Java 8 is a required dependency for Oracle GoldenGate VAM Message Capture.

Using Oracle GoldenGate JMS message capture requires the dynamically linked shared VAM library that is attached to the Oracle GoldenGate Extract process.

10.1.9.3.3 Configuring the VAM Extract

JMS Capture only works with the Oracle GoldenGate Extract process. To run the Java message capture application you need the following:

- Oracle GoldenGate for Java Adapter
- Extract process
- Extract parameter file configured for message capture
- Description of the incoming data format, such as a source definitions file.
- Java 8 installed on the host machine
- [Adding the Extract](#)
- [Configuring the Extract Parameters](#)
- [Configuring Message Capture](#)

10.1.9.3.3.1 Adding the Extract

To add the message capture VAM to the Oracle GoldenGate installation, add an Extract and the trail that it will create using GGSCI commands:

```
ADD EXTRACT jmsvam, VAM
ADD EXTTRAIL dirdat/id, EXTRACT jmsvam, MEGABYTES 100
```

The process name (jmsvam) can be replaced with any process name that is no more than 8 characters. The trail identifier (id) can be any two characters.

Note

Commands to position the Extract, such as BEGIN or EXTRBA, are not supported for message capture. The Extract will always resume by reading messages from the end of the message queue.

10.1.9.3.3.2 Configuring the Extract Parameters

The Extract parameter file contains the parameters needed to define and invoke the VAM. Sample Extract parameters for communicating with the VAM are shown in the table.

Parameter	Description
EXTRACT jmsvam	The name of the Extract process.
VAM ggjava_vam.dll, PARAMS dirprm/jmsvam.properties	Specifies the name of the VAM library and the location of the properties file. The VAM properties should be in the dirprm directory of the Oracle GoldenGate installation location.
TRANLOGOPTIONS VAMCOMPATIBILITY 1	Specifies the original (1) implementation of the VAM is to be used.
TRANLOGOPTIONS GETMETADATAFROMVAM	Specifies that metadata will be sent by the VAM.
EXTTRAIL dirdat/id	Specifies the identifier of the target trail Extract creates.

10.1.9.3.3.3 Configuring Message Capture

Message capture is configured by the properties in the VAM properties file (Adapter Properties file). This file is identified by the PARAMS option of the Extract VAM parameter and used to determine logging characteristics, parser mappings and JMS connection settings.

10.1.9.3.3.4 Connecting and Retrieving the Messages

To process JMS messages you must configure the connection to the JMS interface, retrieve and parse the messages in a transaction, write each message to a trail, commit the transaction, and remove its messages from the queue.

- [Connecting to JMS](#)
- [Retrieving Messages](#)

- [Completing the Transaction](#)

10.1.9.3.4.1 Connecting to JMS

Connectivity to JMS is through a generic JMS interface. Properties can be set to configure the following characteristics of the connection:

- Java classpath for the JMS client
- Name of the JMS queue or topic source destination
- Java Naming and Directory Interface (JNDI) connection properties
 - Connection properties for Initial Context
 - Connection factory name
 - Destination name
- Security information
 - JNDI authentication credentials
 - JMS user name and password

The Extract process that is configured to work with the VAM (such as the `jmsvam` in the example) will connect to the message system. when it starts up.

Note

The Extract may be included in the Manger's AUTORESTART list so it will automatically be restarted if there are connection problems during processing.

Currently the Oracle GoldenGate for Java message capture adapter supports only JMS text messages.

10.1.9.3.4.2 Retrieving Messages

The connection processing performs the following steps when asked for the next message:

- Start a local JMS transaction if one is not already started.
- Read a message from the message queue.
- If the read fails because no message exists, return an end-of-file message.
- Otherwise return the contents of the message.

10.1.9.3.4.3 Completing the Transaction

Once all of the messages that make up a transaction have been successfully retrieved, parsed, and written to the Oracle GoldenGate trail, the local JMS transaction is committed and the messages removed from the queue or topic. If there is an error then the local transaction is rolled back leaving the messages in the JMS queue.

10.1.10 MongoDB

The Oracle GoldenGate capture (Extract) for MongoDB is used to get changes from MongoDB databases.

This chapter describes how to use the Oracle GoldenGate Capture for MongoDB.

- [Overview](#)
- [Prerequisites to Setting up MongoDB](#)
- [MongoDB Database Operations](#)
- [Using Extract Initial Load](#)
- [Using Change Data Capture Extract](#)
- [Positioning the Extract](#)
- [Security and Authentication](#)
- [MongoDB Bidirectional Replication](#)
- [Mongo DB Configuration Reference](#)
- [Columns in Trail File](#)
- [Update Operation Behavior](#)
- [Oplog Size Recommendations](#)
- [Troubleshooting](#)
- [MongoDB Capture Client Dependencies](#)

What are the dependencies for the MongoDB Capture to connect to MongoDB databases?

10.1.10.1 Overview

MongoDB is a document-oriented NoSQL database used for high volume data storage and which provides high performance and scalability along with data modelling and data management of huge sets of data in an enterprise application. MongoDB provides:

- High availability through built-in replication and failover
- Horizontal scalability with native sharding
- End-to-end security and many more

10.1.10.2 Prerequisites to Setting up MongoDB

- MongoDB cluster or a MongoDB node must have a **replica set**. The minimum recommended configuration for a replica set is a three member replica set with three data-bearing members: one primary and two secondary members.

Create mongod instance with the replica set as follows:

```
bin/mongod --bind_ip localhost --port 27017 --replSet rs0 --dbpath ../data/d1/
bin/mongod --bind_ip localhost --port 27018 --replSet rs0 --dbpath ../data/d2/
bin/mongod --bind_ip localhost --port 27019 --replSet rs0 --dbpath ../data/d3/

bin/mongod --host localhost --port 27017
```

Adding a replica set:

```
rs.initiate( {
  _id : "rs0",
```

```
members: [  
  { _id: 0, host: "localhost:27017" },  
  { _id: 1, host: "localhost:27018" },  
  { _id: 2, host: "localhost:27019" }  
]  
})
```

- **Replica Set Oplog**

MongoDB capture uses oplog to read the CDC records. The operations log (oplog) is a capped collection that keeps a rolling record of all operations that modify the data stored in your databases.

The MongoDB only removes an oplog entry in the following cases: the oplog has reached the maximum configured size, and the oplog entry is older than the configured number of hours based on the host system clock.

You can control the retention of oplog entries using: `oplogMinRetentionHours` and `replSetResizeOplog`.

For more information about oplog, see [Oplog Size Recommendations](#).

- You must download and provide the third party libraries of MongoDB clients having version 5.0.0 and forward. See [Reactive Streams Java Driver 5.0.1](#).

Note

MongoDB client version less than 5.0.0 is not supported.

- [Set up Credential Store Entry to Detect Source Type](#)

10.1.10.2.1 Set up Credential Store Entry to Detect Source Type

The database type for capture is based on the prefix in the database credential userid. The generic format for userid is as follows: `<dbtype>://<db-user>@<comma separated list of server addresses>:<port>`. The userid value for MongoDB is any valid MongoDB clientURI without the password.

MongoDB Capture

Example:

```
alter credentialstore add user "mongodb+srv://user@127.0.0.1:27017" password  
db-passwd alias mongo
```

Note

Ensure that the userid value is in double quotes.

MongoDB Atlas

Example:

```
alter credentialstore add user "mongodb+srv://user@127.0.0.1:27017" password  
db-passwd alias mongo
```

10.1.10.3 MongoDB Database Operations

Supported Operations

- INSERT
- UPDATE
- DELETE

Unsupported Operations

The following MongoDB source DDL operations are not supported:

- CREATE collection
- RENAME collection
- DROP collection

On detecting these unsupported operations, extract can be configured to either ABEND or skip these operations and continue processing the next operation.

Note

MongoDB Capture does not include operations related to the admin, config, and local databases.

10.1.10.4 Using Extract Initial Load

MongoDB Extract supports the standard initial load capability to extract source table data to Oracle GoldenGate trail files.

Initial load for MongoDB can be performed to synchronize tables, either as a prerequisite step to replicating changes or as a standalone function.

Configuring the Initial Load

Initial Load Parameter file:

```
-- ggsci> alter credentialstore add user mongodb://db-user@localhost:27017/  
admin password db-passwd alias mongo
```

```
EXTRACT LOAD  
JVMOPTIONS CLASSPATH ggjava/ggjava.jar:/path/to/mongo-capture/libs/*  
SOURCEISTABLE  
SOURCEDB USERIDALIAS mongo  
TABLE database.collection;
```

Run these commands in AdminClient to add extract for initial load:

```
adminclient> ADD EXTRACT load, SOURCEISTABLE  
adminclient> START EXTRACT load
```

10.1.10.5 Using Change Data Capture Extract

Review the example .prm files from Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) installation directory here: AdapterExamples/big-data/mongodbcapture.

When adding the MongoDB Extract trail, you need to use EXTTRAIL to create a local trail file.

The MongoDB Extract trail file should not be configured with the RMTTRAIL option.

```
adminclient> ADD EXTRACT groupname, TRANLOG
adminclient> ADD EXTTRAIL trailprefix, EXTRACT groupname
```

Example:

```
adminclient> ADD EXTRACT mongo, TRANLOG
adminclient> ADD EXTTRAIL ./dirdat/z1, EXTRACT mongo
```

10.1.10.6 Positioning the Extract

MongoDB extract process allows us to position from EARLIEST, TIMESTAMP, EOF and LSN.

EARLIEST: Positions to the start of the Oplog for a given collection.

Syntax:

```
ADD EXTRACT groupname, TRANLOG, EARLIEST
```

TIMESTAMP: Positions to a given time stamp. Token BEGIN can use either NOW to start from present time or with a given timestamp.

```
BEGIN {NOW | yyyy-mm-dd[ hh:mi:[ss[.cccccc]]]}
```

Syntax

```
ADD EXTRACT groupname, TRANLOG, BEGIN NOW
ADD EXTRACT groupname, TRANLOG, BEGIN 'yyyy-mm-dd hh:mm:ss'
```

EOF: Positions to end of oplog.

Syntax

```
ADD EXTRACT groupname, TRANLOG, EOF
```

LSN: Positions to a given LSN.

In MongoDB Capture, the Log Sequence Number (LSN) corresponds to the Operation Time in the oplog, which is unique for each entry. This Operation Time can be represented in two formats: as a timestamp with an increment (in the t.i format) or as a 20-digit numerical value. For example, if the oplog's Operation Time is "ts": {"\$timestamp": {"t": 1733328879, "i": 2}}, the corresponding LSN can be expressed as 1733328879.2 or 07444590848517341186.

The syntax for adding an extract is as follows:

For the timestamp.increment format:

```
ADD EXTRACT groupname, TRANLOG, LSN  
"1733328879.2"
```

For the 20-digit LSN format:

```
ADD EXTRACT groupname, TRANLOG, LSN "07444590848517341186"
```

10.1.10.7 Security and Authentication

Oracle GoldenGate (OGG) provides robust mechanisms for securely connecting to MongoDB using credential stores, encrypted authentication, and TLS/mTLS encryption.

- [Credential Store Integration](#)
- [MongoDB URI-Based Authentication](#)
MongoDB uses a URI-style connection string that supports various authentication methods and options as query parameters.
- [SSL Configuration Setup](#)

10.1.10.7.1 Credential Store Integration

Oracle GoldenGate supports a secure credential store where MongoDB credentials are stored encrypted. This mechanism:

- Eliminates the need to hard-code usernames and passwords in parameter files.
- Stores connection URIs securely using an alias.
- References credentials using the alias in all configuration.

Setup using AdminClient

- adminclient> add credentialstore
- adminclient> alter credentialstore add user "mongodb://
myUserAdmin@localhost:27017/admin?replicaSet=rs0" password
MySecurePassword alias mongo

Note

The MongoDB URI must be enclosed in double quotes.

Logging in with the Alias

- adminclient> dblogin useridalias mongo
- adminclient> list tables tcust*

Use alias in parameter file

```
SOURCEDB USERIDALIAS mongo
```

10.1.10.7.2 MongoDB URI-Based Authentication

MongoDB uses a URI-style connection string that supports various authentication methods and options as query parameters.

- [Authentication Methods](#)
- [TLS/SSL Configuration](#)

10.1.10.7.2.1 Authentication Methods

- **SCRAM-SHA-256** : Recommended password-based authentication mechanism.
Example: `mongodb://username@host:27017/?authSource=admin&authMechanism=SCRAM-SHA-256`
- **SCRAM-SHA-1** : For backward compatibility.
Example: `mongodb://username@host:27017/?authSource=admin&authMechanism=SCRAM-SHA-1`
- **PLAIN (LDAP)**: Requires TLS.
Example: `mongodb://ldapUser@host:27017/?authMechanism=PLAIN&authSource=$external&tls=true`
- **x.509 Certificate Authentication**:
Example: `mongodb://host:27017/?authMechanism=MONGODB-X509&tls=true&tlsCertificateKeyFile=client.pem&tlsCAFile=ca.pem`

Note

For more information on X509 configuration steps, see [MongoDB Extract and Replicat Configuration](#).

10.1.10.7.2.2 TLS/SSL Configuration

MongoDB connections can be secured using TLS or mutual TLS (mTLS).

Note

The MongoDB extract utilizes both the C driver and the Java driver of MongoDB. Therefore, it is essential to respect the `tlsCAFile` and `tlsCertificateKeyFile` parameters in the `clientURL` when establishing a `DBConnection`, as well as to include the `keystore/truststore` details in the JVM bootoptions during the extract process. For more information, see [SSL Configuration Setup](#) to generate the certificates required for `tls`/`mtls`.

- [Disable TLS \(Not Recommended\)](#)
- [Enable TLS](#)
- [Mutual TLS \(mTLS\)](#)

10.1.10.7.2.2.1 Disable TLS (Not Recommended)

Example: `mongodb://host:27017/?tls=false`

10.1.10.7.2.2.2 Enable TLS

- If server uses the CA-trusted certificate:
Example: `mongodb://host:27017/?tls=true`
- If server uses the self-signed certificate:

Example: `mongodb://host:27017/?
tls=true&tlsAllowInvalidCertificates=true`

or

Example: `mongodb://host:27017/?tls=true&tlsCAFile=/path/to/ca.pem`

Note

The truststore information should be provided in the JVM bootoptions when used `tlsCAFile` in `clientURI`.

10.1.10.7.2.2.3 Mutual TLS (mTLS)

Performs the client certificate validation along with the server certificate validation.

Example: `mongodb://host:27017/?tls=true&tlsCAFile=/path/to/
ca.pem&tlsCertificateKeyFile=/path/to/client.pem`

Note

The truststore and keystore information should be provided in the JVM bootoptions.

JVM bootoptions for mTLS:

- `-Djavax.net.ssl.trustStore=/path/to/truststore.jks`
- `-Djavax.net.ssl.trustStorePassword=changeit`
- `-Djavax.net.ssl.keyStore=/path/to/keystore.jks`
- `-Djavax.net.ssl.keyStorePassword=changeit`

10.1.10.7.3 SSL Configuration Setup

To configure SSL between the MongoDB instance and Oracle GoldenGate for Distributed Applications and Analytics MongoDB Capture, do the following:

Create certificate authority (CA)

```
openssl req -passout pass:password -new -x509 -days 3650 -extensions v3_ca -  
keyout  
ca_private.pem -out ca.pem -subj  
"/CN=CA/OU=GOLDENGATE/O=ORACLE/L=BANGALORE/ST=KA/C=IN"
```

Create key and certificate signing requests (CSR) for client and all server nodes

```
openssl req -newkey rsa:4096 -nodes -out client.csr -keyout client.key -subj  
'/CN=certName/OU=OGGBDCLIENT/O=ORACLE/L=BANGALORE/ST=AP/C=IN'  
openssl req -newkey rsa:4096 -nodes -out server.csr -keyout server.key -subj  
'/CN=slc13auo.us.oracle.com/OU=GOLDENGATE/O=ORACLE/L=BANGALORE/ST=TN/C=IN'
```

Sign the certificate signing requests with CA

```
openssl x509 -passin pass:password -sha256 -req -days 365 -in client.csr -CA
ca.pem -CAkey
ca_private.pem -CAcreateserial -out client-signed.crtopenssl x509 -passin
pass:password -sha256 -req -days 365 -in server.csr -CA ca.pem -CAkey
ca_private.pem -CAcreateserial -out server-signed.crt -extensions v3_req -
extfile
<(cat << EOF[ v3_req ]subjectAltName = @alt_names
[ alt_names ]
DNS.1 = 127.0.0.1
DNS.2 = localhost
DNS.3 = hostname
EOF)
```

Create the privacy enhanced mail (PEM) file for mongod

```
cat client-signed.crt client.key > client.pem
cat server-signed.crt server.key > server.pem
```

Create trust store and keystore

```
openssl pkcs12 -export -out server.pkcs12 -in server.pem
openssl pkcs12 -export -out client.pkcs12 -in client.pem

bash-4.2$ ls
ca.pem ca_private.pem client.csr client.pem server-signed.crt
server.key server.pkcs12
ca.srl client-signed.crt client.key client.pkcs12 server.csr
server.pem
```

Start instances of mongod with the following options:

```
--tlsMode requireTLS --tlsCertificateKeyFile ../opensslKeys/server.pem --
tlsCAFile
    ../opensslKeys/ca.pem
```

credentialstore connectionString

```
alter credentialstore add user
    mongodb://myUserAdmin@localhost:27017/admin?
ssl=true&tlsCertificateKeyFile=../mcopysslkeys/
client.pem&tlsCertificateKeyFilePassword=password&tlsCAFile=../mcopysslkeys/
ca.pem
    password root alias mongo
```

Note

The Length of connectionString should not exceed 256.

For CDC Extract, add the key store and trust store as part of the JVM options.

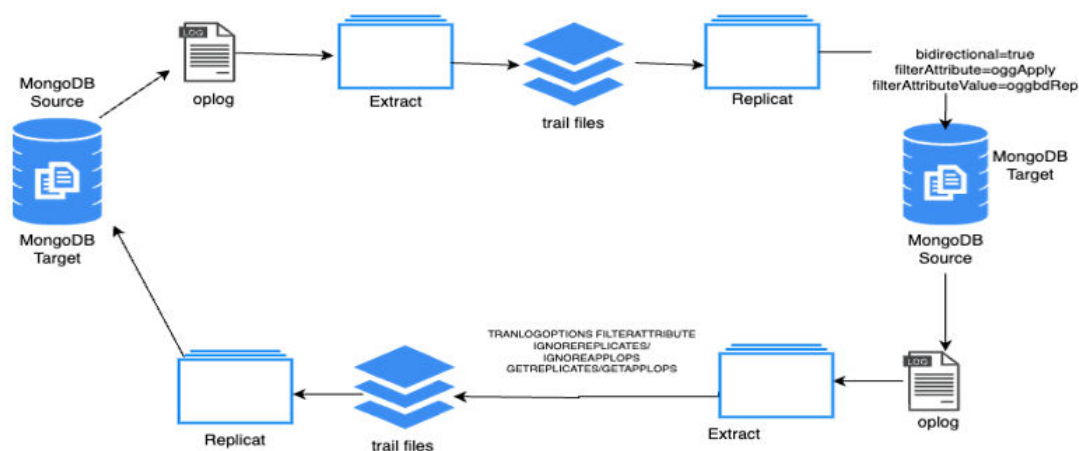
JVM options

```
-Xms512m -Xmx4024m -Xss32m -Djavax.net.ssl.trustStore=../mcpensslkeys /
server.pkcs12
-Djavax.net.ssl.trustStorePassword=password
-Djavax.net.ssl.keyStore =../mcpensslkeys/client.pkcs12
-Djavax.net.ssl.keyStorePassword=password
```

10.1.10.8 MongoDB Bidirectional Replication

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) has integration to capture changes from a MongoDB source database, and also apply the changes to a MongoDB target database. In bidirectional replication, Changes that are made to one source collection are replicated to target collection, and changes that are made to the second copy are replicated back to the first copy.

This topic explains the design to support bidirectional replication for MongoDB.



① Note

MongoDB Version 6 or above is required to support bi-directional replication. With versions before 6.0, MongoDB bi-directional is not supported and it fails with the following error message: *MONGODB-000XX MongoDB version should be 6 or greater to support bi-directional replication.*

- [Enabling Trandata](#)
- [Enabling MongoDB Bi-directional Replication](#)
- [Extracting from Target Replicat which is Bidirectionally Processed](#)
- [Troubleshooting](#)

10.1.10.8.1 Enabling Trandata

Before starting the replicat process with bidirectional enabled, one should enable the trandata for the collection where the data is been replicated. By enabling the trandata on the collection before the start of the replicat process, will capture the before image of the operation with

which an Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) extract process can identify if the document is processed by the GG for DAA or not.

Extract abends if trandata is not enabled on the collection that been used in the bidirectional enabled replicat process.

Command to Enable Trandata

```
Dblogin useridalias <aliasname>  
"add trandata <schema>.<collectionname>"
```

Note

The target collection should be available before the replicat process when executed with bidirectionally enabled.

10.1.10.8.2 Enabling MongoDB Bi-directional Replication

To enable MongoDB bi-directional replication, set `gg.handler.mongodb.bidirectional` to `true` (`gg.handler.mongodb.bidirectional=true`) in replicat properties.

When `gg.handler.mongodb.bidirectional` property is set to `true`, replicat process adds `filterAttribute` and `filterAttributeValue` key value pair to the document. `filterAttribute` and `filterAttributeValue` is needed for loop-detection. Ensure that the `filterAttributeValue` contain only ASCII characters [A-Za-z] and numbers [0-9] with a Maximum length of 256 characters. If the document has the key-value pair of `filterAttribute` and `filterAttributeValue`, then it shows that the document is processed by Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) replicat process.

When `gg.handler.mongodb.bidirectional` property is set to `true`, replicat ingests the default value of `filterAttribute` as `oggApply` and the default `filterAttributeValue` as `true` if not specified explicitly. You can enable MongoDB bi-directional replication with default settings. For example: `gg.handler.mongodb.bidirectional=true`

```
{ "_id" : ObjectId("65544aa60b0a066d021ba508"), "CUST_CODE" : "test65",  
  "name" : "hello  
    world", "cost" : 3000, "oggApply":"true"}
```

You can also define the key-value pair of `filterAttribute` and `filterAttributeValue`. For example:

```
gg.handler.mongodb.bidirectional=true  
gg.handler.mongodb.filterAttribute=region  
gg.handler.mongodb.filterAttributeValue=westcentral
```

Sample insert doc with custom key-value pair:

```
{ "_id" : ObjectId("65544aa60b0a066d021ba508"), "CUST_CODE" : "test65",  
  "name" : "hello world", "cost" : 3000, "region":"westcentral"}
```

10.1.10.8.3 Extracting from Target Replicat which is Bidirectionally Processed

TRANLOGOPTIONS EXCLUDEFILTERATTRIBUTE can be used in extract parameters to filter source MongoDB operations. TRANLOGOPTIONS EXCLUDEFILTERATTRIBUTE is a value/ key pair. Default EXCLUDEFILTERATTRIBUTE attribute name and value is oggApply and true. Optionally, name and value can be set by user. User can mention multiple TRANLOGOPTIONS EXCLUDEFILTERATTRIBUTE options with different key value pairs.

This option may be used to avoid data looping in a bidirectional configuration of MongoDB capture by specifying EXCLUDEFILTERATTRIBUTE name with the value that was used by MongoDB Replicat.

Example 1

TRANLOGOPTIONS EXCLUDEFILTERATTRIBUTE filters attribute with oggApply and value with true. If the source document contains the specified EXCLUDEFILTERATTRIBUTE, the document will be filtered and will not be extracted.

TRANLOGOPTIONS EXCLUDEFILTERATTRIBUTE

Filtered Sample Message:

```
{ "_id" : ObjectId("65544aa60b0a066d021ba508"), "CUST_CODE" : "test65",
  "name" : "hello world", "cost" : 3000, "oggApply":"true"}
```

TRANLOGOPTIONS EXCLUDEFILTERATTRIBUTE parameter value should be in line with source replicat's FILTERATTRIBUTE and FILTERATTRIBUTEVALUE to detect the loop or decide to process/ filter the operations. If the source document contains the specified FILTERATTRIBUTE, the document is identified as a replicated operation.

Example 2

**

The following extract parameter filters the replicated operations marked with attribute region and value westcentral. And captures the application operations. Also, if there are other operations marked with a different attribute value, they will be extracted.

TRANLOGOPTIONS EXCLUDEFILTERATTRIBUTE region=westcentral

Filtered sample message:

```
{ "_id" : ObjectId("65544aa60b0a066d021ba508"), "CUST_CODE" : "test65", "name" :
  "hello world", "cost" : 3000, "region":"westcentral"}
```

Extracted sample message:

```
{ "_id" : ObjectId("1881aa60bMKA66d021b1938"), "CUST_CODE" : "test38",
  "name" : "hello world", "cost" : 2000 }
```

**

Note

From version 23.4 onwards, the extract parameter FILTERATTRIBUTE is renamed to EXCLUDEFILTERATTRIBUTE, the parameters GETREPLICATES/IGNOREREPLICATE and GETAPPLOPS/IGNOREAPPLOPS are deprecated. Usage of these parameters results in abend of the extract process.

10.1.10.8.4 Troubleshooting

1. In bidirectional replication, If no before image is available for the delete document then abend the process and error out.

Sample error

MONGODB-000XX No before image is available for collection [<collection name>] with the document [<document>].

2. If MongoDB version used is less than 6, then MONGODB-000XX MongoDB version should be 6 or greater to support bi-directional replication.

10.1.10.9 Mongo DB Configuration Reference

The following properties are used with MongoDB change data capture.

Notes

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
				p l i c a t i o n s a n d A n a l y t i c s (G G f o r D A A) r e l e a s e 2 3 . 4 . 0 . 0 . 0 , t h

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
				i s p a r a m e t e r w i l l b e d e p r e c a t e d .
JVMOPTIONS [CLASSPATH <classpath> BOOTOPTIONS <options>]	Optio nal	Extract Paramet er file	None	CLASSPATH: The classpath for the Java Virtual Machine. You can include an asterisk (*) wildcard to match all JAR files in any directory. Multiple paths should be delimited with a colon (:) character. BOOTOPTIONS: The boot options for the Java Virtual Machine. Multiple options are delimited by a space character.

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
JVMB00TOPTIONS <i>jvm_options</i>	Optio nal	GLOBA LS file	None	The boot options for the Java Virtual Machine. Multiple options are delimited by a space character.

 Note
Starting from Oracle Database 12c, the JVM options for GoldenGate are listed in the `GGSCI` command line interface.

Properties	Req uire d/ Opti onal	Locatio n	Default	Explanation
				parameter will be deprecated.

Properties	Required/Optional	Location	Default	Explanation
JVMCLASSPATH <classpath>	Required	Global file	None	The classpath for the Java Virtual Machine. You can include an asterisk (*) wildcard to match all JAR files in any directory. Multiple paths should be delimited with a colon (:) character. Example: <pre>JVMCLASSPATH ggjava/ggjava.jar:/path/to/ mongodb_client_dependencyjars/*</pre>

Properties	Required/ Optional	Location	Default	Explanation
parameter				
SOURCEDB USERIDALIAS < <i>alias name</i> >	Required	Extract parameter (.prm) file	None	This parameter is used by the extract process for authentication in to the source MongoDB database. The <i>alias name</i> refers to the alias that should exist in Oracle Wallet. See Security and Authentication .
ABEND_ON_DDL	Optional	CDC Extract parameter (.prm) file	None	This is a default behaviour of MongoDB Capture extract. On detection of CREATE collection, RENAME collection, and DROP collection, extract process will be abended.
NO_ABEND_ON_DDL	Optional	CDC Extract parameter (.prm) file	None	On detection of CREATE collection, RENAME collection, and DROP collection, extract process skips these operations and continue processing the next operation.
ABEND_ON_DROP_DATABASE	Optional	CDC Extract parameter (.prm) file	None	This is a default behaviour of MongoDB Capture extract. On detection of Drop Database operation, extract process will be abended.

Properties	Required/ Optional	Location	Default	Explanation
NO_ABEND_ON_DROP_DATABASE	Optional	CDC Extract parameter (.prm) file.	None	On detection of Drop Database operation, extract process will skip these operations and continue processing the next operation.
BINARY_JSON_FORMAT	Optional	prm	None	When configured BINARY_JSON_FORMAT, MongoDB Capture process represents documents in BSON format, and using BINARY_JSON_FORMAT is more performance efficient. If BINARY_JSON_FORMAT is not specified, then documents are represented in Extended JSON format which is human-readable and less performance efficient compared to using BINARY_JSON_FORMAT. When using BINARY_JSON_FORMAT - in the generated trail file, the column metadata has data_type as 64, sub_data_type as 4, and Remarks as JSON. When BINARY_JSON_FORMAT is not specified - in the generated trail file, the column metadata has data_type as 64, sub_data_type as 0, and Remarks as JSON. For more information, see Table Metadata.
TRANLOGOPTIONS FETCHPARTIALJSON	Optional	CDC Extract parameter (.prm) file	None	On configuring tranlogoptions FETCHPARTIALJSON, the extract process does a DB lookup and fetches the full document for the given update operation. See MongoDB Bidirectional Replication .
TRANLOGOPTIONS READFROMCHANGESTREAM	Optional	CDC Extract parameter (.prm) file	None	On configuring TRANLOGOPTIONS READFROMCHANGESTREAM, Capture monitor documents through a Change Stream in DocumentDB.
DBOPTIONS JSON_MODE <CANONICAL/RELAXED>	Optional	CDC Extract parameter (.prm) file	CANONICAL	Specifies the JSON output format for MongoDB payload data. Valid values are: RELAXED: Human-readable format that represents BSON types using standard JSON where possible. CANONICAL: Strict format that preserves type fidelity by representing all BSON types explicitly.

Table Metadata

When BINARY_JSON_FORMAT is configured, the column metadata should have data_type as 64, sub_data_type as 4, and JSON as the Remarks.

Example:

```
2021/11/11 06:45:06.311.849 Metadata          Len 143 RBA 1533
Table Name: MYTEST.TEST
```

```

*
1)Name          2)Data Type          3)External Length  4)Fetch Offset
5)Scale          6)Level
7)Null           8)Bump if Odd        9)Internal Length 10)Binary Length   11)Table
Length 12)Most Sig DT
13)Least Sig DT 14)High Precision 15)Low Precision 16)Elementary Item
17)Occurs       18)Key Column
19)Sub DataType 20)Native DataType 21)Character Set 22)Character Length 23)LOB
Type           24)Partial Type
25)Remarks
*
TDR version: 11
Definition for table MYTEST.TEST
Record Length: 16010
Columns: 2
id          64  8000          0 0 0 0 0  8000  8000          0 0 0 0 0 1  0 1  4
-4          -1    0 0 0  JSON
payload     64  8000          8005 0 0 1 0  8000  8000          0 0 0 0 0 1  0 0    4
-4          -1    0 1 0  JSON
End of definition
s

```

When BINARY_JSON_FORMAT is not configured, the column metadata should have data_type as 64, sub_data_type as 0, and JSON as the Remarks.

Example

```

2021/11/11 06:45:06.311.849 Metadata          Len 143 RBA 1533
Table Name: MYTEST.TEST
*
1)Name          2)Data Type          3)External Length  4)Fetch Offset
5)Scale          6)Level
7)Null           8)Bump if Odd        9)Internal Length 10)Binary Length   11)Table
Length 12)Most Sig DT
13)Least Sig DT 14)High Precision 15)Low Precision 16)Elementary Item
17)Occurs       18)Key Column
19)Sub DataType 20)Native DataType 21)Character Set 22)Character Length 23)LOB
Type           24)Partial Type
25)Remarks
*
TDR version: 11
Definition for table MYTEST.TEST
Record Length: 16010
Columns: 2
id          64  8000          0 0 0 0 0  8000  8000          0 0 0 0 0 1  0 1  0
-4          -1    0 0 0  JSON
payload     64  8000          8005 0 0 1 0  8000  8000          0 0 0 0 0 1  0 0  0
-4          -1    0 1 0  JSON
End of definition

```

10.1.10.10 Columns in Trail File

Each trail records will have two columns:

- Column 0 as ‘_id’, which identifies a document in a collection.
- Column 1 as ‘payload’, which holds all the columns (fields of a collection).

Based on property BINARY_JSON_FORMAT, columns are presented as a BSON format or Extended JSON format. When BINARY_JSON_FORMAT is configured, the captured documents are represented in the BSON format as follows.


```

2021/10/26 06:21:33.000.000 Insert                               Len   329 RBA 1921
Name: MYTEST.TEST (TDR Index: 1)
After Image:
Partition x0c G s
0000 1a00 0000 1600 1600 0000 075f 6964 0061 7800 | .....ax.
ddc2 d894 d2f5 fca4 9e00 0100 2701 0000 2301 2301 | .....#.#.
0000 075f 6964 0061 7800 ddc2 d894 d2f5 fca4 9e02 | ..._id.ax.....
4355 5354 5f43 4f44 4500 0500 0000 7361 6162 0002 | CUST_CODE.....saab..
6e61 6d65 0005 0000 006a 6f68 6e00 026c 6173 746e | name.....john..lastn
616d 6500 0500 0000 7769 6c6c 0003 6164 6472 6573 | ame.....will..adres
7365 7300 8300 0000 0373 7472 6565 7464 6574 6169 | ses.....streetdetai
Column 0 (0x0000), Length 26 (0x001a) id.
0000 1600 1600 0000 075f 6964 0061 7800 ddc2 d894 | .....ax.....
d2f5 fca4 9e00 | .....
Column 1 (0x0001), Length 295 (0x0127) payload.
0000 2301 2301 0000 075f 6964 0061 7800 ddc2 d894 | ..#.....ax.....
d2f5 fca4 9e02 4355 5354 5f43 4f44 4500 0500 0000 | .....CUST_CODE.....
7361 6162 0002 6e61 6d65 0005 0000 006a 6f68 6e00 | saab..name.....john.
026c 6173 746e 616d 6500 0500 0000 7769 6c6c 0003 | .lastname.....will..
6164 6472 6573 7365 7300 8300 0000 0373 7472 6565 | addresses.....stree
7464 6574 6169 6c73 006f 0000 0003 6172 6561 0020 | tdetails.o.....area.
0000 0003 5374 7265 6574 0013 0000 0001 6c61 6e65 | ....Street.....lane
0000 0000 0000 005e 4000 0003 666c 6174 6465 7461 | .....^@...flatdeta
696c 7300 3700 0000 0166 6c61 746e 6f00 0000 0000 | ils.7....flatno.....
0040 6940 0270 6c6f 746e 6f00 0300 0000 3262 0002 | .@i@.plotno.....2b..
6c61 6e65 0009 0000 0032 6e64 7068 6173 6500 0000 | lane.....2ndphase...
0003 7072 6f76 6973 696f 6e00 3000 0000 0373 7461 | ..provision.0....sta
7465 0024 0000 0003 6b61 001b 0000 0002 6b61 726e | te.$....ka.....karn
6174 616b 6100 0700 0000 3537 3031 3032 0000 0000 | ataka.....570102....
0263 6974 7900 0400 0000 626c 7200 00 | .city.....blr..

```

When BINARY_JSON_FORMAT is not configured, the captured documents are represented in the JSON format as follows:

```

2021/10/01 01:09:35.000.000 Insert                               Len   366 RBA 1711
Name: MYTEST.testarr (TDR Index: 1)
After Image:
Partition x0c G s
0000 2700 0000 2300 7b22 246f 6964 223a 2236 3135 | ..'...#..{"$oid":"615
3663 3233 6633 3466 3061 3965 3661 3735 3536 3930 | 6c23f34f0a9e6a755690
6422 7d01 003f 0100 003b 017b 225f 6964 223a 207b | d"}..?...;..{"_id": {
2224 6f69 6422 3a20 2236 3135 3663 3233 6633 3466 | "$oid": "6156c23f34f
3061 3965 3661 3735 3536 3930 6422 7d2c 2022 4355 | 0a9e6a755690d"}, "CU
5354 5f43 4f44 4522 3a20 2265 6d70 3122 2c20 226e | ST_CODE": "emp1", "n
616d 6522 3a20 226a 6f68 6e22 2c20 226c 6173 746e | ame": "john", "lastn
Column 0 (0x0000), Length 39 (0x0027).
0000 2300 7b22 246f 6964 223a 2236 3135 3663 3233 | ..#..{"$oid":"6156c23
6633 3466 3061 3965 3661 3735 3536 3930 6422 7d | f34f0a9e6a755690d"}
Column 1 (0x0001), Length 319 (0x013f).
0000 3b01 7b22 5f69 6422 3a20 7b22 246f 6964 223a | ..;..{"_id": {"$oid":
2022 3631 3536 6332 3366 3334 6630 6139 6536 6137 | "6156c23f34f0a9e6a7
3535 3639 3064 227d 2c20 2243 5553 545f 434f 4445 | 55690d"}, "CUST_CODE
223a 2022 656d 7031 222c 2022 6e61 6d65 223a 2022 | ": "emp1", "name": "
6a6f 686e 222c 2022 6c61 7374 6e61 6d65 223a 2022 | john", "lastname": "
7769 6c6c 222c 2022 6164 6472 6573 7365 7322 3a20 | will", "addresses":
7b22 7374 7265 6574 6465 7461 696c 7322 3a20 7b22 | {"streetdetails": {"
6172 6561 223a 207b 2253 7472 6565 7422 3a20 7b22 | area": {"Street": {"
6c61 6e65 223a 2031 3230 2e30 7d7d 2c20 2266 6c61 | lane": 120.0}}, "fla
7464 6574 6169 6c73 223a 207b 2266 6c61 746e 6f22 | tdetails": {"flatno"
3a20 3230 322e 302c 2022 706c 6f74 6e6f 223a 2022 | : 202.0, "plotno": "
3262 222c 2022 6c61 6e65 223a 2022 326e 6470 6861 | 2b", "lane": "2ndpha
7365 227d 7d7d 2c20 2270 726f 7669 7369 6f6e 223a | se}}}}, "provision":
207b 2273 7461 7465 223a 207b 226b 6122 3a20 7b22 | {"state": {"ka": {"
6b61 726e 6174 616b 6122 3a20 2235 3730 3130 3222 | karnataka": "570102"

```

```
7d7d 7d2c 2022 6369 7479 223a 2022 626c 7222 7d | }}, "city": "blr"}
```

10.1.10.11 Update Operation Behavior

MongoDB Capture extract reads change records from the capped collection `oplog.rs`. For Update operations, the collection contains information on the modified fields only. Thus the MongoDB Capture extract will write only the modified fields in trail on Update operation as MongoDB native `$set` and `$unset` documents.

Example trail record:

```
2022/02/22 01:26:52.000.000 FieldComp          Len   243 RBA 1711
Name: lobt.MNGUPSRT (TDR Index: 1)
Min. Replicat version: 21.5, Min. GENERIC version: 0.0, Incompatible Replicat: Abend
Column 0 (0x0000), Length 55 (0x0037) id.
0000 3300 7b20 225f 6964 2220 3a20 7b20 2224 6f69 | ..3. { "_id" : { "$oi
6422 203a 2022 3632 3133 3633 3064 3931 3561 6631 | d" : "6213630d915af1
3633 3265 6264 6461 3766 2220 7d20 7d          | 632ebdda7f" } }
Column 1 (0x0001), Length 180 (0x00b4) payload.
0000 b000 7b22 2476 223a 207b 2224 6e75 6d62 6572 | .... { "$v" : { "$number
496e 7422 3a20 2231 227d 2c20 2224 7365 7422 3a20 | Int": "1"}, "$set":
7b22 6c61 7374 4d6f 6469 6669 6564 223a 207b 2224 | {"lastModified": {"$
6461 7465 223a 207b 2224 6e75 6d62 6572 4c6f 6e67 | date": {"$numberLong
223a 2022 3136 3435 3532 3230 3132 3238 3522 7d7d | ": "1645522012285"}}
2c20 2273 697a 652e 756f 6d22 3a20 2263 6d22 2c20 | , "size.uom": "cm",
2273 7461 7475 7322 3a20 2250 227d 2c20 225f 6964 | "status": "P"}, "_id
223a 207b 2224 6f69 6422 3a20 2236 3231 3336 3330 | ": {"$oid": "6213630
6439 3135 6166 3136 3332 6562 6464 6137 6622 7d7d | d915af1632ebdda7f"}}
```

GGs tokens:

```
TokenID x50 'P' COLPROPERTY          Info x01 Length   6
Column:    1, Property: 0x02, Remarks: Partial
TokenID x74 't' ORATAG                Info x01 Length   0
TokenID x4c 'L' LOGCSN                Info x00 Length  20
3037 3036 3734 3633 3232 3633 3838 3131 3935 3533 | 07067463226388119553
TokenID x36 '6' TRANID                Info x00 Length  19
3730 3637 3436 3332 3236 3338 3831 3139 3535 33   | 7067463226388119553
```

Here The GGS token x50 with Remarks as Partial indicates that this record is a partial record.

On configuring `tranlogoptions FETCHPARTIALJSON`, the extract process does a database lookup and fetches the full document for the given update operation.

Example

```
2022/02/22 01:26:59.000.000 FieldComp          Len   377 RBA 2564
Name: lobt.MNGUPSRT (TDR Index: 1)
Column 0 (0x0000), Length 55 (0x0037) id.
0000 3300 7b20 225f 6964 2220 3a20 7b20 2224 6f69 | ..3. { "_id" : { "$oi
6422 203a 2022 3632 3133 3633 3064 3931 3561 6631 | d" : "6213630d915af1
3633 3265 6264 6461 3764 2220 7d20 7d          | 632ebdda7d" } }
Column 1 (0x0001), Length 314 (0x013a) payload.
0000 3601 7b20 225f 6964 2220 3a20 7b20 2224 6f69 | ..6. { "_id" : { "$oi
6422 203a 2022 3632 3133 3633 3064 3931 3561 6631 | d" : "6213630d915af1
3633 3265 6264 6461 3764 2220 7d2c 2022 6974 656d | 632ebdda7d" }, "item
2220 3a20 226d 6f75 7365 7061 6422 2c20 2271 7479 | " : "mousepad", "qty
2220 3a20 7b20 2224 6e75 6d62 6572 446f 7562 6c65 | " : { "$numberDouble
2220 3a20 2232 352e 3022 207d 2c20 2273 697a 6522 | " : "25.0" }, "size"
203a 207b 2022 6822 203a 207b 2022 246e 756d 6265 | : { "h" : { "$numbe
7244 6f75 626c 6522 203a 2022 3139 2e30 2220 7d2c | rDouble" : "19.0" },
```

```

2022 7722 203a 207b 2022 246e 756d 6265 7244 6f75 | "w" : { "$numberDou
626c 6522 203a 2022 3232 2e38 3530 3030 3030 3030 | ble" : "22.850000000
3030 3030 3031 3432 3122 207d 2c20 2275 6f6d 2220 | 000001421" }, "uom"
3a20 2269 6e22 207d 2c20 2273 7461 7475 7322 203a | : "in" }, "status" :
2022 5022 2c20 226c 6173 744d 6f64 6966 6965 6422 | "P", "lastModified"
203a 207b 2022 2464 6174 6522 203a 207b 2022 246e | : { "$date" : { "$n
756d 6265 724c 6f6e 6722 203a 2022 3136 3435 3532 | umberLong" : "164552
3230 3139 3936 3122 207d 207d 207d | 2019961" } } }

```

GGs tokens:

```

TokenID x46 'F' FETCHEDDATA      Info x01 Length 1
6                                  | Current by key
TokenID x4c 'L' LOGCSN           Info x00 Length 20
3037 3036 3734 3633 3235 3634 3532 3839 3036 3236 | 07067463256452890626
TokenID x36 '6' TRANID           Info x00 Length 19
3730 3637 3436 3332 3536 3435 3238 3930 3632 36   | 7067463256452890626

```

Here The GGS token x46 FETCHEDDATA indicates that this record is full image for the update operation.

10.1.10.12 Oplog Size Recommendations

By default, MongoDB uses 5% of disk space as oplog size.

Oplog should be long enough to hold all transactions for the longest downtime you expect on a secondary. At a minimum, an oplog should be able to hold minimum 72 hours of operations or even a week's work of operations.

Before mongod creates an oplog, you can specify its size with the `--oplogSize` option.

After you have started a replica set member for the first time, use the `replSetResizeOplog` administrative command to change the oplog size. `replSetResizeOplog` enables you to resize the oplog dynamically without restarting the mongod process.

Workloads Requiring Larger Oplog Size

If you can predict your replica set's workload to resemble one of the following patterns, then you might want to create an oplog that is larger than the default. Conversely, if your application predominantly performs reads with a minimal amount of write operations, a smaller oplog may be sufficient.

The following workloads might require a larger oplog size.

Updates to Multiple Documents at Once

The oplog must translate multi-updates into individual operations in order to maintain idempotency. This can use a great deal of oplog space without a corresponding increase in data size or disk use.

Deletions Equal the Same Amount of Data as Inserts

If you delete roughly the same amount of data as you insert, then the database doesn't grow significantly in disk use, but the size of the operation log can be quite large.

Significant Number of In-Place Updates

If a significant portion of the workload is updates that do not increase the size of the documents, then the database records a large number of operations but does not change the quantity of data on disk.

10.1.10.13 Troubleshooting

- **Error : com.mongodb.MongoQueryException: Query failed with error code 11600 and error message 'interrupted at shutdown' on server localhost:27018.**
The MongoDB server is killed or closed. Restart the Mongod instances and MongoDB capture.
- **Error: java.lang.IllegalStateException: state should be: open.**
The active session is closed due to the session's idle time-out value getting exceeded. Increase the mongod instance's `logicalSessionTimeoutMinutes` parameter value and restart the Mongod instances and MongoDB capture.
- **Error:Exception in thread "main" com.mongodb.MongoQueryException: Query failed with error code 136 and error message 'CollectionScan died due to position in capped collection being deleted. Last seen record id: RecordId(6850088381712443337)' on server localhost:27018 at com.mongodb.internal.operation.QueryHelper.translateCommandException(QueryHelper.java:29)**
This Exception happens when we have Fast writes to mongod and insufficient oplog size. See [Oplog Size Recommendations](#).
- **Error: not authorized on DB to execute command**
This error occurs due to insufficient privileges for the user. The user must be authenticated to run the specified command.
- **Error: com.mongodb.MongoClientException: Sessions are not supported by the MongoDB cluster to which this client is connected.**
Ensure that the Replica Set is available and accessible. In case of MongoDB instance migration from a different version, set the property `FeatureCompatibilityVersion` as follows:


```
db.adminCommand( { setFeatureCompatibilityVersion: "3.6" } ){_}
```
- **Error: bson_append_array(): Invalid array detected. First element of array parameter is not "0".**
To suppress the given message in the Amazon DocumentDB capture report file, append the query param `"&authMechanism=SCRAM-SHA-1"` to the connection URL.

10.1.10.14 MongoDB Capture Client Dependencies

What are the dependencies for the MongoDB Capture to connect to MongoDB databases?

Oracle GoldenGate requires that you use the 5.x MongoDB reactive streams or higher integration with MongoDB. You can download this driver from: <https://search.maven.org/artifact/org.mongodb/mongodb-driver-reactivestream>

- [Reactive Streams Java Driver 5.0.1](#)

10.1.10.14.1 Reactive Streams Java Driver 5.0.1

The required dependent client libraries are: `bson.jar`, `mongodb-driver-core.jar`, `mongodb-driver-reactivestreams.jar`, and `reactive-streams.jar` and `reactor-core.jar`

You must include the path to the MongoDB reactivestreams Java driver in the `gg.classpath` property. To automatically download the Java driver from the Maven central repository, add the

following Maven coordinates of these third party libraries that are needed to run MongoDB Change Data Capture in the pom.xml file:

```
<!-- https://search.maven.org/artifact/org.mongodb/mongodb-driver-reactivestreams
-->
<dependency>
<groupId>org.mongodb</groupId>
<artifactId>mongodb-driver-reactivestreams</artifactId>
<version>5.0.1</version>
</dependency>

<dependency>
<groupId>org.mongodb</groupId>
<artifactId>bson</artifactId>
<version>5.0.1</version>
</dependency>

<dependency>
<groupId>org.mongodb</groupId>
<artifactId>mongodb-driver-core</artifactId>
<version>5.0.1</version>
</dependency>

<dependency>
<groupId>org.reactivestreams</groupId>
<artifactId>reactive-streams</artifactId>
<version>1.0.4</version>
</dependency>

<dependency>
<groupId>io.projectreactor</groupId>
<artifactId>reactor-core</artifactId>

<version>3.5.0</version>
</dependency>
```

The jars are as follows:

- bson-5.1.0.jar
- bson-record-codec-5.1.0.jar
- mongodb-driver-core-5.1.0.jar
- mongodb-driver-reactivestreams-5.1.0.jar
- reactive-streams-1.0.4.jar
- reactor-core-3.5.0.jar

Download version 5.1.0 from Maven central at: <https://mvnrepository.com/artifact/org.mongodb/mongodb-driver-reactivestreams>.

10.1.11 Microsoft Fabric Eventstreams

This chapter describes how to use the Microsoft Fabric eventstreams .

- [Overview](#)
- [Prerequisites](#)

10.1.11.1 Overview

Microsoft Fabric eventstreams is a feature in Microsoft Fabric that captures, transforms, and routes real-time events to various destinations. You can add event data sources, destinations, and transformations to the eventstream. Microsoft Fabric eventstreams enables the event ingestion without writing custom code.

You can create an eventstream, add event data sources, and route to supported destinations. Microsoft Fabric eventstreams exposes a Kafka-compatible endpoint that allows Oracle GoldenGate Kafka Extract to consume events using the standard Apache Kafka protocol. For more information about configuring Oracle GoldenGate Kafka Extract, see [Apache Kafka](#) as a source.

10.1.11.2 Prerequisites

Before configuring Oracle GoldenGate for Microsoft Fabric eventstream:

1. Create a Microsoft Fabric eventstream with a custom endpoint destination.
2. Retrieve the Kafka bootstrap servers, SASL credentials and Topic name from the custom endpoint (under SAS Key Authentication).
3. Ensure network accessibility from the Oracle GoldenGate deployment to the eventstream Kafka endpoint.

Following is a sample consumer .properties to consume from Microsoft Fabric eventstream:

```
bootstrap.servers={YOUR_BOOTSTRAP_SERVERS}
group.id=oggFabricConsumerGroup
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule
required.username="$ConnectionString" password="{YOUR_CONNECTION_STRING}";
```

Note

Ensure credentials and bootstrap server settings are correctly fetched from the Microsoft Fabric eventstream UI or API.

Related Topics

- [Microsoft Fabric eventstreams](#)

10.1.12 OCI Streaming

To capture messages from OCI Streaming and parse into logical change records with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA), you can use Kafka Extract. For more information, see [Apache Kafka](#) as source.

10.2 Target

GoldenGate for Distributed Applications and Analytics (GG for DAA) supports the following technologies as source. GG for DAA change data replication is managed by a replicat process.

About Replicat

Replicat is a process that delivers data to a target system. It reads the source trail file, reconstructs the DML or DDL operations, and applies them to the target system.

For the following two common uses cases of GG for DAA, the function of the Replicat process is as follows:

- **Initial Loads:** When you set up GG for DAA for initial loads, the Replicat process applies a static data copy to target objects or routes the data to a high-speed bulk-load utility
- **Change Synchronization:** When you set up GG for DAA to keep the target system synchronized with the source, the Replicat process applies the source operations to the target objects.

You can configure multiple Replicat processes with one or more Extract processes to increase the throughput. To preserve data integrity, each set of processes handles a different set of objects. To differentiate among Replicat processes, you assign each one a group name.

- [Amazon DocumentDB](#)
- [Add Replicat](#)
- [Amazon Kinesis](#)
The Kinesis Streams Handler streams data to applications hosted on the Amazon Cloud or in your environment.
- [Amazon MSK](#)
- [Amazon Redshift](#)
Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. The purpose of the Redshift Event Handler is to apply operations into Redshift tables.
- [Amazon S3](#)
Learn how to use the S3 Event Handler, which provides the interface to Amazon S3 web services.
- [Apache Cassandra](#)
The Cassandra Handler provides the interface to Apache Cassandra databases.
- [Apache HBase](#)
The HBase Handler is used to populate HBase tables from existing Oracle GoldenGate supported sources.
- [Apache HDFS](#)
The HDFS Handler is designed to stream change capture data into the Hadoop Distributed File System (HDFS).
- [Apache Kafka](#)
The Kafka Handler is designed to stream change data from an Oracle GoldenGate trail to a Kafka topic.
- [Apache Hive](#)
- [Apache Iceberg](#)
- [Azure Blob Storage](#)
- [Azure Data Lake Storage](#)

- [Azure Event Hubs](#)
Kafka handler supports connectivity to Microsoft Azure Event Hubs.
- [Azure Synapse Analytics Data Warehouse](#)
Microsoft Azure Synapse Analytics is a limitless analytics service that brings together data integration, enterprise data warehousing and Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) analytics.
- [Databricks](#)
- [DataStax](#)
- [Confluent Kafka](#)
- [Elasticsearch](#)
- [Flat Files](#)
- [Google BigQuery](#)
- [Google Cloud Storage](#)
- [Google Cloud Managed Service for Apache Kafka](#)
The Google Cloud Managed Service for Apache Kafka is a fully managed, Apache Kafka-compatible service on Google Cloud .
- [Google Pub/Sub](#)
- [Java Message Service \(JMS\)](#)
The Java Message Service (JMS) Handler allows operations from a trail file to be formatted in messages, and then published to JMS providers like Oracle Weblogic Server, Websphere, and ActiveMQ.
- [Java Database Connectivity](#)
Learn how to use the Java Database Connectivity (JDBC) Handler, which can replicate source transactional data to a target or database.
- [Microsoft Fabric Eventstreams](#)
This chapter describes how to use the Microsoft Fabric eventstreams.
- [Microsoft Fabric OneLake](#)
- [MongoDB](#)
Learn how to use the MongoDB Handler, which can replicate transactional data from Oracle GoldenGate to a target MongoDB and Autonomous JSON databases (AJD and ATP) .
- [Oracle AI Data Platform](#)
- [OCI Streaming](#)
Oracle Cloud Infrastructure Streaming (OCI Streaming) supports putting messages to and receiving messages using the Kafka client. Therefore, Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) can be used to publish change data capture operation messages to OCI Streaming.
- [Oracle NoSQL](#)
The Oracle NoSQL Handler can replicate transactional data from Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) to a target Oracle NoSQL Database.
- [OCI Autonomous Data Warehouse](#)
Oracle Autonomous Data Warehouse (ADW) is a fully managed database tuned and optimized for data warehouse workloads with the market-leading performance of Oracle Database.

- [OCI Object Storage](#)
The Oracle Cloud Infrastructure (OCI) Event Handler is used to load files generated by the File Writer Handler into an Oracle Cloud Infrastructure Object Store.
- [Redis](#)
Redis is an in-memory data structure store which supports optional durability. Redis is simply a key/value data store where a unique key identifies the data structure stored. The value is the data structure that is stored.
- [Snowflake](#)
- [Additional Details](#)
- [Oracle GoldenGate Java Delivery](#)

10.2.1 Amazon DocumentDB

Amazon DocumentDB is a fully-managed database service and it is compatible with MongoDB drivers and MongoDB APIs.

For more information about replicating data to Amazon DocumentDB, see [MongoDB](#).

All the configuration steps are similar to the ones supported for [MongoDB](#), except the TLS configuration.

To support TLS, you need to generate the appropriate keystore files, see the **Java** section at the following: https://docs.aws.amazon.com/documentdb/latest/developerguide/connect_programmatically.html#connect_programmatically-tls_enabled.

After the truststore is generated, it can be configured under the replicat .props file as follows:

```
jvm.bootoptions=-Djavax.net.ssl.trustStore={insert the path and trust store file name}  
-Djavax.net.ssl.trustStorePassword={insert the trust store password}
```

Note

Target AWS DocumentDB access requires GG for DAA deployment to be in the same VPC or a different VPC configured with VPC peering.

10.2.2 Add Replicat

Replicat can be configured in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 26ai web-based user interface.

Note

Complete the prerequisites based on source technology type.

To create a Replicat, refer the following section for technology-specific details:

1. Download dependencies using dependency downloader for the source technology and note the path to dependency files.
2. If required, create the credential store entry.
3. In GG for DAA UI, go to **Administration Service** and click **Add Replicat**.

4. In **Replicat Information**, select a **Replicat Type** and provide a Process Name for the Replicat.
5. In **Replicat Options**, provide a name for **Replicat Trail**, select **Target** from drop-down list. Depending on the target selected, replicat properties are auto populated in the **Properties File**.
6. The **Managed Options** are optional, enable if needed.

Option	Description
Profile Name	Provides the name of the autostart and autorestart profile. You can select the default or custom options. If you have already created a profile, then you can select that profile also. If you select the Custom option, then you can set up a new profile from this section itself.
Critical to deployment health	(Oracle only) Enable this option if the profile is critical for the deployment health. Note: This option only appears while creating the Extract or Replicat and not when you set up the managed processes in the Profiles page.
Auto Start	Enables autostart for the process
Startup Delay	Time to wait in seconds before starting the process
Auto Restart	Configures how to restart the process if it terminates
Max Retries	Specify the maximum number of retries to try to start the process
Retry Delay	Delay time in trying to start the process
Retries Window	The duration interval to try to start the process
Restart on Failure only	If true, the task is only restarted if it fails.
Disable Task After Retries Exhausted	If true, then the task is disabled after exhausting all attempts to restart the process.

7. On the **Parameters File** screen, enter the basic parameters for setting up a Replicat. By default, the Parameter File text contains some parameters, which can be customized as required.
If Coordinated Replicat is used, add TARGETDB LIBFILE libggjava.so SET property=/u02/Deployment/etc/conf/ogg/your_replicat_name.properties to parameters file.
8. On the **Properties File** screen, update the properties marked as **TODO**. In gg.classpath, provide the path to dependency files you downloaded in previous step using dependency downloader for the source technology and noted the path to dependency file.
9. Click **Create and Run** to start the Replicat after adding the replicat.
10. Click **Create** to add the Replicat.

 Note

Do not start the replicat immediately after being created.

Release 26ai: For more information about replication options, see the subsequent sections for technology-specific details.

10.2.3 Amazon Kinesis

The Kinesis Streams Handler streams data to applications hosted on the Amazon Cloud or in your environment.

This chapter describes how to use the Kinesis Streams Handler.

- [Overview](#)
- [Detailed Functionality](#)
- [Kinesis Handler Performance Considerations](#)
- [Troubleshooting](#)

10.2.3.1 Overview

Amazon Kinesis is a messaging system that is hosted in the Amazon Cloud. Kinesis streams can be used to stream data to other Amazon Cloud applications such as Amazon S3 and Amazon Redshift. Using the Kinesis Streams Handler, you can also stream data to applications hosted on the Amazon Cloud or at your site. Amazon Kinesis streams provides functionality similar to Apache Kafka.

The logical concepts map is as follows:

- Kafka Topics = Kinesis Streams
- Kafka Partitions = Kinesis Shards

A Kinesis stream must have at least one shard.

10.2.3.2 Detailed Functionality

- [Amazon Kinesis Java SDK](#)
- [Kinesis Streams Input Limits](#)

10.2.3.2.1 Amazon Kinesis Java SDK

The Oracle GoldenGate Kinesis Streams Handler uses the AWS Kinesis Java SDK to push data to Amazon Kinesis, see *Amazon Kinesis Streams Developer Guide* at:

<http://docs.aws.amazon.com/streams/latest/dev/developing-producers-with-sdk.html>.

The Kinesis Streams Handler was designed and tested with the latest AWS Kinesis Java SDK version 2.28.11. These are the dependencies:

- Group ID: `software.amazon.awssdk`
- Artifact ID: `kinesis`
- Version: `2.28.11`

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) does not ship with the AWS Kinesis Java SDK. Oracle recommends that you use the AWS Kinesis Java SDK identified in the Certification Matrix, see [GoldenGate Certifications](#).

Note

It is assumed by moving to the latest AWS Kinesis Java SDK that there are no changes to the interface, which can break compatibility with the Kinesis Streams Handler.

You can download the AWS Java SDK, including Kinesis from:

<https://aws.amazon.com/sdk-for-java/>

10.2.3.2.2 Kinesis Streams Input Limits

The upper input limit for a Kinesis stream with a single shard is 1000 messages per second up to a total data size of 1MB per second. Adding streams or shards can increase the potential throughput such as the following:

- 1 stream with 2 shards = 2000 messages per second up to a total data size of 2MB per second
- 3 streams of 1 shard each = 3000 messages per second up to a total data size of 3MB per second

The scaling that you can achieve with the Kinesis Streams Handler depends on how you configure the handler. Kinesis stream names are resolved at runtime based on the configuration of the Kinesis Streams Handler.

Shards are selected by the hash the partition key. The partition key for a Kinesis message cannot be null or an empty string (""). A null or empty string partition key results in a Kinesis error that results in an abend of the Replicat process.

Maximizing throughput requires that the Kinesis Streams Handler configuration evenly distributes messages across streams and shards.

To achieve the best distribution across shards in a Kinesis stream, select a partitioning key which rapidly changes. You can select `${primaryKeys}` as it is unique per row in the source database. Additionally, operations for the same row are sent to the same Kinesis stream and shard. When the DEBUG logging is enabled, the Kinesis stream name, sequence number, and the shard number are logged to the log file for successfully sent messages.

10.2.3.3 Kinesis Handler Performance Considerations

- [Kinesis Streams Input Limitations](#)
- [Transaction Batching](#)
- [Deferring Flush at Transaction Commit](#)

10.2.3.3.1 Kinesis Streams Input Limitations

The maximum write rate to a Kinesis stream with a single shard to be 1000 messages per second up to a maximum of 1MB of data per second. You can scale input to Kinesis by adding additional Kinesis streams or adding shards to streams. Both adding streams and adding shards can linearly increase the Kinesis input capacity and thereby improve performance of the Oracle GoldenGate Kinesis Streams Handler.

Adding streams or shards can linearly increase the potential throughput such as follows:

- 1 stream with 2 shards = 2000 messages per second up to a total data size of 2MB per second.
- 3 streams of 1 shard each = 3000 messages per second up to a total data size of 3MB per second.

To fully take advantage of streams and shards, you must configure the Oracle GoldenGate Kinesis Streams Handler to distribute messages as evenly as possible across streams and shards.

Adding additional Kinesis streams or shards does nothing to scale Kinesis input if all data is sent to using a static partition key into a single Kinesis stream. Kinesis streams are resolved at runtime using the selected mapping methodology. For example, mapping the source table name as the Kinesis stream name may provide good distribution of messages across Kinesis streams if operations from the source trail file are evenly distributed across tables. Shards are selected by a hash of the partition key. Partition keys are resolved at runtime using the selected mapping methodology. Therefore, it is best to choose a mapping methodology to a partition key that rapidly changes to ensure a good distribution of messages across shards.

10.2.3.3.2 Transaction Batching

The Oracle GoldenGate Kinesis Streams Handler receives messages and then batches together messages by Kinesis stream before sending them via synchronous HTTPS calls to Kinesis. At transaction commit all outstanding messages are flushed to Kinesis. The flush call to Kinesis impacts performance. Therefore, deferring the flush call can dramatically improve performance.

The recommended way to defer the flush call is to use the GROUPTRANSOPS configuration in the replicat configuration. The GROUPTRANSOPS groups multiple small transactions into a single larger transaction deferring the transaction commit call until the larger transaction is completed. The GROUPTRANSOPS parameter works by counting the database operations (inserts, updates, and deletes) and only commits the transaction group when the number of operations equals or exceeds the GROUPTRANSOPS configuration setting. The default GROUPTRANSOPS setting for replicat is 1000.

Interim flushes to Kinesis may be required with the GROUPTRANSOPS setting set to a large amount. An individual call to send batch messages for a Kinesis stream cannot exceed 500 individual messages or 5MB. If the count of pending messages exceeds 500 messages or 5MB on a per stream basis then the Kinesis Handler is required to perform an interim flush.

10.2.3.3.3 Deferring Flush at Transaction Commit

The messages are by default flushed to Kinesis at transaction commit to ensure write durability. However, it is possible to defer the flush beyond transaction commit. This is only advisable when messages are being grouped and sent to Kinesis at the transaction level (that is one transaction = one Kinesis message or chunked into a small number of Kinesis messages), when the user is trying to capture the transaction as a single messaging unit.

This may require setting the GROUPTRANSOPS replication parameter to 1 so as not to group multiple smaller transactions from the source trail file into a larger output transaction. This can impact performance as only one or few messages are sent per transaction and then the transaction commit call is invoked which in turn triggers the flush call to Kinesis.

In order to maintain good performance the Oracle GoldenGate Kinesis Streams Handler allows the user to defer the Kinesis flush call beyond the transaction commit call. The Oracle GoldenGate replicat process maintains the checkpoint in the .cpr file in the {GoldenGate Home}/dirchk directory. The Java Adapter also maintains a checkpoint file in this directory named .cpj. The Replicat checkpoint is moved beyond the checkpoint for which the Oracle

GoldenGate Kinesis Handler can guarantee message loss will not occur. However, in this mode of operation the GoldenGate Kinesis Streams Handler maintains the correct checkpoint in the .cpj file. Running in this mode will not result in message loss even with a crash as on restart the checkpoint in the .cpj file is parsed if it is before the checkpoint in the .cpr file.

10.2.3.4 Troubleshooting

- [Java Classpath](#)
- [Kinesis Handler Connectivity Issues](#)
- [Logging](#)

10.2.3.4.1 Java Classpath

The most common initial error is an incorrect classpath to include all the required AWS Kinesis Java SDK client libraries and creates a `ClassNotFoundException` in the log file.

You can troubleshoot by setting the Java Adapter logging to `DEBUG`, and then rerun the process. At the debug level, the logging includes information about which JARs were added to the classpath from the `gg.classpath` configuration variable.

The `gg.classpath` variable supports the wildcard asterisk (*) character to select all JARs in a configured directory. For example, `/usr/kinesis/sdk/*`, see [Setting Up and Running the Kinesis Streams Handler](#).

10.2.3.4.2 Kinesis Handler Connectivity Issues

If the Kinesis Streams Handler is unable to connect to Kinesis when running on premise, the problem can be the connectivity to the public Internet is protected by a proxy server. Proxy servers act a gateway between the private network of a company and the public Internet. Contact your network administrator to get the URLs of your proxy server, and then follow the directions in [Configuring the Proxy Server for Kinesis Streams Handler](#).

10.2.3.4.3 Logging

The Kinesis Streams Handler logs the state of its configuration to the Java log file.

This is helpful because you can review the configuration values for the handler. Following is a sample of the logging of the state of the configuration:

```
**** Begin Kinesis Streams Handler - Configuration Summary ****
Mode of operation is set to op.
  The AWS region name is set to [us-west-2].
  A proxy server has been set to [www-proxy.us.oracle.com] using port [80].
  The Kinesis Streams Handler will flush to Kinesis at transaction commit.
  Messages from the GoldenGate source trail file will be sent at the operation level.
  One operation = One Kinesis Message
The stream mapping template of [${fullyQualifiedTableName}] resolves to [fully qualified
table name].
The partition mapping template of [${primaryKeys}] resolves to [primary keys].
**** End Kinesis Streams Handler - Configuration Summary ****
```

10.2.4 Amazon MSK

Amazon MSK is a fully managed, secure, and a highly available Apache Kafka service. You can use [Apache Kafka](#) to replicate to Amazon MSK.

10.2.5 Amazon Redshift

Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. The purpose of the Redshift Event Handler is to apply operations into Redshift tables.

See [Flat Files](#).

- [Detailed Functionality](#)
Ensure to use the Redshift Event handler as a downstream Event handler connected to the output of the S3 Event handler. The S3 Event handler loads files generated by the File Writer Handler into Amazon S3.
- [Operation Aggregation](#)
- [Unsupported Operations and Limitations](#)
- [Uncompressed UPDATE records](#)
The uncompressed UPDATE operation records contain full image of the row being updated. If the **Delete Insert** configuration or the **Aggregate Operations** configuration is set to true in the properties file, it is mandatory that the trail files used to apply to Redshift should contain the uncompressed UPDATE operation records.
- [Error During the Data Load Proces](#)
Staging operation data from AWS S3 onto temporary staging tables and updating the target table occurs inside a single transaction. In case of any error(s), the entire transaction is rolled back and the replicat process will ABEND.
- [Troubleshooting and Diagnostics](#)
- [Classpath](#)
Redshift apply relies on the upstream File Writer handler and the S3 Event handler.
- [Configuration](#)
- [INSERTALLRECORDS Support](#)
- [Redshift COPY SQL Authorization](#)
The Redshift event handler uses COPY SQL to read staged files in Amazon Web Services (AWS) S3 buckets. The COPY SQL query may need authorization credentials to access files in AWS S3.
- [Co-ordinated Apply Support](#)
- [Support for Mixed Case Identifiers](#)
- [AWS Secrets Manager Authentication](#)
- [Restrictions on Database Object Names](#)

10.2.5.1 Detailed Functionality

Ensure to use the Redshift Event handler as a downstream Event handler connected to the output of the S3 Event handler. The S3 Event handler loads files generated by the File Writer Handler into Amazon S3.

Redshift Event handler uses the COPY SQL to bulk load operation data available in S3 into temporary Redshift staging tables. The staging table data is then used to update the target table. All the SQL operations are performed in batches providing better throughput.

10.2.5.2 Operation Aggregation

- [In-Memory Operation Aggregation](#)

- [Aggregation using SQL post loading data into the staging table](#)
In this aggregation operation, the in-memory operation aggregation need not be performed. The operation data loaded into the temporary staging table is aggregated using SQL queries, such that the staging table contains just one row per key.

10.2.5.2.1 In-Memory Operation Aggregation

- Operation records are aggregated in-memory by default.
- The `gg.aggregate.operations.flush.interval` property has been deprecated and is no longer supported. If the `gg.aggregate.operations.flush.interval` is used in GG for DAA 26ai, then replicat will run; but add a warning to log file about the property being deprecated and not supported.
To control the time window for aggregation, use the `gg.handler.redshift.fileRollInterval` property. By default, it is set to 3 minutes. Longer intervals will increase latency, and may increase memory usage. Shorter intervals will increase overhead in Oracle GoldenGate and the target database.
- Operation aggregation in-memory requires additional JVM memory configuration.

10.2.5.2.2 Aggregation using SQL post loading data into the staging table

In this aggregation operation, the in-memory operation aggregation need not be performed. The operation data loaded into the temporary staging table is aggregated using SQL queries, such that the staging table contains just one row per key.

Table 10-3 Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.aggregate.operations.using.sql=true</code>	Optional	True False	False	Use SQL to aggregate staging table data before updating the target table.

10.2.5.3 Unsupported Operations and Limitations

The following operations are not supported by the Redshift Handler:

- DDL changes are not supported.
- Timestamp and Timestamp with Time zone data types: The maximum precision supported is up to microseconds, the nanoseconds portion will be truncated. This is a limitation we have observed with the Redshift COPY SQL.
- Redshift COPY SQL has a limitation on the maximum size of a single input row from any source is 4MB.

10.2.5.4 Uncompressed UPDATE records

The uncompressed UPDATE operation records contain full image of the row being updated. If the **Delete Insert** configuration or the **Aggregate Operations** configuration is set to true in the properties file, it is mandatory that the trail files used to apply to Redshift should contain the uncompressed UPDATE operation records.

If UPDATE records have missing columns, then such columns are updated in the target as null. By setting the parameter `gg.abend.on.missing.columns=true`, replicat can fail fast on detecting a compressed update trail record. This is the recommended setting.

10.2.5.5 Error During the Data Load Proces

Staging operation data from AWS S3 onto temporary staging tables and updating the target table occurs inside a single transaction. In case of any error(s), the entire transaction is rolled back and the replicat process will ABEND.

If there are errors with the COPY SQL, then the Redshift system table `stl_load_errors` is also queried and the error traces are made available in the handler log file.

10.2.5.6 Troubleshooting and Diagnostics

- Connectivity issues to Redshift
 - Validate JDBC connection URL, user name and password.
 - Check if http/https proxy is enabled. Generally, Redshift endpoints cannot be accessed via proxy.
- DDL and Truncate operations not applied on the target table: The Redshift handler will ignore DDL and truncate records in the source trail file.
- Target table existence: It is expected that the Redshift target table exists before starting the apply process. Target tables need to be designed with primary keys, sort keys, partition distribution key columns. Approximations based on the column metadata in the trail file may not be always correct. Therefore, Redshift apply will ABEND if the target table is missing.
- Operation aggregation in-memory (`gg.aggregate.operations=true`) is memory intensive where as operation aggregation using `SQL(gg.eventhandler.name.aggregateStagingTableRows=true)` requires more SQL processing on the Redshift database. These configurations are mutually exclusive and only one of them should be enabled at a time. Tests within Oracle have revealed that operation aggregation in memory delivers better apply rate. This may not always be the case on all the customer deployments.
- Diagnostic information on the apply process is logged onto the handler log file.
 - Operation aggregation time (in milli-seconds) in-memory:

```
INFO 2018-10-22 02:53:57.000980 [pool-5-thread-1] - Merge statistics
*****START*****
INFO 2018-10-22 02:53:57.000980 [pool-5-thread-1] - Number of update
operations merged into an existing update operation: [232653]
INFO 2018-10-22 02:53:57.000980 [pool-5-thread-1] - Time spent aggregating
operations : [22064]
INFO 2018-10-22 02:53:57.000980 [pool-5-thread-1] - Time spent flushing
aggregated operations : [36382]
INFO 2018-10-22 02:53:57.000980 [pool-5-thread-1] - Merge statistics
*****END*****
```

- Stage and load processing time (in milli-seconds) for SQL queries

```
INFO 2018-10-22 02:54:19.000338 [pool-4-thread-1] - Stage and load statistics
*****START*****
INFO 2018-10-22 02:54:19.000338 [pool-4-thread-1] - Time spent for staging
process [277093]
INFO 2018-10-22 02:54:19.000338 [pool-4-thread-1] - Time spent for load
process [32650]
INFO 2018-10-22 02:54:19.000338 [pool-4-thread-1] - Stage and load statistics
*****END*****
```

- Stage time (in milli-seconds) will also include additional statistics if operation aggregation using SQL is enabled.
- Co-existence of the components: The location/region of the machine where replicat process is running, AWS S3 bucket region and the Redshift cluster region would impact the overall throughput of the apply process. Data flow is as follows: GoldenGate => AWS S3 => AWS Redshift. For best throughput, the components need to be located as close as possible.

10.2.5.7 Classpath

Redshift apply relies on the upstream File Writer handler and the S3 Event handler.

Include the required jars needed to run the S3 Event handler in gg.classpath. See [Amazon S3](#). Redshift Event handler uses the Redshift JDBC driver. Ensure to include the jar file in gg.classpath as shown in the following example:

```
gg.classpath=aws_sdk_2.28.11/lib/:aws_sdk_2.28.11/third-party/lib/./redshift-jdbc42-2.1.0.29.jar
```

10.2.5.8 Configuration

Automatic Configuration

AWS Redshift Data warehouse replication involves configuring of multiple components, such as file writer handler, S3 event handler and Redshift event handler. The Automatic Configuration feature auto configures these components so that you need to perform minimal configurations. The properties modified by auto configuration will also be logged in the handler log file.

To enable auto configuration to replicate to Redshift target, set the parameter:
gg.target=redshift

```
gg.target
Required
Legal Value: redshift
Default: None
Explanation: Enables replication to Redshift target
```

When replicating to Redshift target, the customization of S3 event handler name and Redshift event handler name is not allowed.

File Writer Handler Configuration

File writer handler name is pre-set to the value redshift. The following is an example to edit a property of file writer handler: `gg.handler.redshift.pathMappingTemplate=./dirout`

S3 Event Handler Configuration

S3 event handler name is pre-set to the value s3. The following is an example to edit a property of the S3 event handler: `gg.eventhandler.s3.bucketMappingTemplate=bucket1`.

Redshift Event Handler Configuration

The Redshift event handler name is pre-set to the value redshift.

Table 10-4 Properties

Properties	Required/Optional	Legal Value	Default	Explanation
<code>gg.eventhandler.redshift.connectionURL</code>	Required	Redshift JDBC Connection URL	None	Sets the Redshift JDBC connection URL. Example: <code>jdbc:redshift://aws-redshift-instance.cjoaij3df5if.us-east-2.redshift.amazonaws.com:5439/mydb</code>
<code>gg.eventhandler.redshift.UserName</code>	Required	JDBC User Name	None	Sets the Redshift database user name.
<code>gg.eventhandler.redshift.Password</code>	Required	JDBC Password	None	Sets the Redshift database password.
<code>gg.eventhandler.redshift.awsIamRole</code>	Optional	AWS role ARN in the format: <code>arn:aws:iam::<aws_account_id>:role/<role_name></code>	None	AWS IAM role ARN that the Redshift cluster uses for authentication and authorization for executing COPY SQL to access objects in AWS S3 buckets.
<code>gg.eventhandler.redshift.useAwsSecurityTokenService</code>	Optional	true false	Value is set from the configuration property set in the upstream s3 Event handler <code>gg.eventhandler.s3.enableSTS</code>	Use AWS Security Token Service for authorization. For more information, see Redshift COPY SQL Authorization .

Table 10-4 (Cont.) Properties

Properties	Required/Optional	Legal Value	Default	Explanation
gg.eventhandler .redshift.awsSTSEndpoint	Optional	A valid HTTPS URL.	Value is set from the configuration property set in the upstream s3 Event handler gg.eventhandler.s3.stsURL.	The AWS STS endpoint string. For example: https://sts.us-east-1.amazonaws.com . For more information, see Redshift COPY SQL Authorization .
gg.eventhandler .redshift.awsSTSRegion	Optional	A valid AWS region.	Value is set from the configuration property set in the upstream s3 Event handler gg.eventhandler.s3.stsRegion.	The AWS STS region. For example, us-east-1. For more information, see Redshift COPY SQL Authorization .
gg.initialLoad	Optional	true false	false	If set to true, initial load mode is enabled. See INSERTALLRECORDS Support .
gg.eventhandler .redshift.connectionRetryIntervalSeconds	Optional	Integer Value	30	Specifies the delay in seconds between connection retry attempts.
gg.eventhandler .redshift.connectionRetries	Optional	Integer Value	3	Specifies the number of times connections to the target data warehouse will be retried.

Table 10-4 (Cont.) Properties

Properties	Required/Optional	Legal Value	Default	Explanation
gg.handler.redshift.fileRollInterval	Optional	The default unit of measure is milliseconds. You can stipulate ms, s, m, h to signify milliseconds, seconds, minutes, or hours respectively. Examples of legal values include 10000, 10000ms, 10s, 10m, or 1.5h. Values of 0 or less indicate that file rolling on time is turned off.	3m (three minutes)	The parameter determines how often the data will be merged into Redshift. Use with caution, the higher this value is the more data will need to be stored in the memory of the Replicat process.

Note
Use the parameter name to review with caution on .inc read session

Table 10-4 (Cont.) Properties

Properties	Required/Optional	Legal Value	Default	Explanation

g
i
t
s
d
e
f
a
u
l
t
v
a
l
u
e
(
3
m
)
w
i
l
l
i
n
c
r
e
a
s
e
t
h
e
a
m
o
u
n
t
o
f
d
a
t
a
s
t
o
r
e
d
i

Table 10-4 (Cont.) Properties

Properties	Required/Optional	Legal Value	Default	Explanation

n
t
h
e
i
n
t
e
r
n
a
l
m
e
m
o
r
y
o
f
t
h
e
R
e
p
l
i
c
a
t
.
T
h
i
s
c
a
n
c
a
u
s
e
o
u
t
o
f
m
e
m
o
r

Table 10-4 (Cont.) Properties

Properties	Required/Optional	Legal Value	Default	Explanation

your error standards stop the Replicator from running out of memory.

Table 10-4 (Cont.) Properties

Properties	Required/Optional	Legal Value	Default	Explanation

t
e
S
t
a
r
t
i
n
g
w
i
t
h
t
h
e
2
3
a
i
r
e
l
e
a
s
e
.
t
h
e
g
g
.
a
g
g
r
e
g
a
t
e
.
o
p
e
r
a
t
i

Table 10-4 (Cont.) Properties

Properties	Required/Optional	Legal Value	Default	Explanation

ons
s
.
f
l
u
s
h
.
i
n
t
e
r
v
a
l
p
r
o
p
e
r
t
y
i
s
d
e
p
r
e
c
a
t
e
d
a
n
d
n
o
l
o
n
g
e
r
s
u
p
p

Table 10-4 (Cont.) Properties

Properties	Required/Optional	Legal Value	Default	Explanation

Oracle GoldenGate for Distributed Applications and Analytics
May 29, 2026
Page 113 of 771

Table 10-4 (Cont.) Properties

Properties	Required/Optional	Legal Value	Default	Explanation
gg.eventhandler .snowflake.deleteInsert	Optional	True or False	False	If set to true, Replicat will merge records using SQLDELETE+INSERT statements instead of SQL MERGE statement.
gg.eventhandler .redshift.secretsManagerSecretId	Optional	AWS Secrets Manager secret id	None	Set this to retrieve the JDBC username and password from an AWS Secrets Manager secret. gg.eventhandler.redshift.secretsManagerRegion must be set as well, otherwise, the replicat will abend. Properties gg.eventhandler.redshift.UserName and gg.eventhandler.redshift.Password are no longer required when this is set.
gg.eventhandler .redshift.secretsManagerRegion	Optional	A valid AWS region	None	The AWS Secrets Manager region. For example, us-east-1.

End-to-End Configuration

The following is an end-end configuration example which uses auto configuration for FW handler, S3 and Redshift Event handlers.

The sample properties are available at the following location:

- In an Oracle GoldenGate Classic install: <oggbd_install_dir>/AdapterExamples/big-data/redshift-via-s3/rs.props
- In an Oracle GoldenGate Microservices install: <oggbd_install_dir>/opt/AdapterExamples/big-data/redshift-via-s3/rs.props

```
# Configuration to load GoldenGate trail operation records
# into Amazon Redshift by chaining
# File writer handler -> S3 Event handler -> Redshift Event handler.
# Note: Recommended to only edit the configuration marked as TODO
```

```

gg.target=redshift
#The S3 Event Handler
#TODO: Edit the AWS region
gg.eventhandler.s3.region=<aws region>
#TODO: Edit the AWS S3 bucket
gg.eventhandler.s3.bucketMappingTemplate<s3bucket>

#The Redshift Event Handler
#TODO: Edit ConnectionUrl
gg.eventhandler.redshift.connectionURL=jdbc:redshift://aws-redshift-
instance.cjoaij3df5if.us-east-2.redshift.amazonaws.com:5439/mydb
#TODO: Edit Redshift user name
gg.eventhandler.redshift.UserName=<db user name>
#TODO: Edit Redshift password
gg.eventhandler.redshift.Password=<db password>
#TODO:Set the classpath to include AWS Java SDK and Redshift JDBC driver.
gg.classpath=aws_sdk_2.28.11/lib/:aws_sdk_2.28.11/third-party/lib/./redshift-
jdbc42-2.1.0.29.ja

```

10.2.5.9 INSERTALLRECORDS Support

Stage and merge targets supports INSERTALLRECORDS parameter.

See [INSERTALLRECORDS](#) in *Reference for Oracle GoldenGate*. Set the INSERTALLRECORDS parameter in the Replicat parameter file (.prm).

Setting this property directs the Replicat process to use bulk insert operations to load operation data into the target table. You can tune the batch size of bulk inserts using the File Writer property `gg.handler.redshift.maxFileSize`. The default value is set to 1GB. The frequency of bulk inserts can be tuned using the File Writer property `gg.handler.redshift.fileRollInterval`, the default value is set to 3m (three minutes).

Note

10.2.5.10 Redshift COPY SQL Authorization

The Redshift event handler uses COPY SQL to read staged files in Amazon Web Services (AWS) S3 buckets. The COPY SQL query may need authorization credentials to access files in AWS S3.

Authorization can be provided by using an AWS Identity and Access Management (IAM) role that is attached to the Redshift cluster or by providing a AWS access key and a secret for the access key. As a security consideration, it is a best practise to use role-based access when possible.

AWS Key-Based Authorization

With key-based access control, you provide the access key ID and secret access key for an AWS IAM user that is authorized to access AWS S3. The access key id and secret access key are retrieved by looking up the credentials as follows:

1. Environment variables - `AWS_ACCESS_KEY/AWS_ACCESS_KEY_ID` and `AWS_SECRET_KEY/AWS_SECRET_ACCESS_KEY`.
2. Java System Properties - `aws.accessKeyId` and `aws.secretAccessKey`.
3. Credential profiles file at the default location (`~/.aws/credentials`).

4. Amazon Elastic Container Service (ECS) container credentials loaded from Amazon ECS if the environment variable `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` is set.
5. Instance profile credentials retrieved from Amazon Elastic Compute Cloud (EC2) metadata service.

Running Replicat on an AWS EC2 Instance

If the replicat process is started on an AWS EC2 instance, then the access key ID and secret access key are automatically retrieved by Oracle GoldenGate for BigData and no explicit user configuration is required.

Temporary Security Credentials using AWS Security Token Service (STS)

If you use the key-based access control, then you can further limit the access users have to your data by retrieving temporary security credentials using AWS Security Token Service. The auto configure feature of the Redshift event handler automatically picks up the AWS Security Token Service (STS) configuration from S3 event handler.

Table 10-5 S3 Event Handler Configuration and Redshift Event Handler Configuration

S3 Event Handler Configuration	Redshift Event Handler Configuration
<code>enableSTS</code>	<code>useAwsSTS</code>
<code>stsURL</code>	<code>awsSTSEndpoint</code>
<code>stsRegion</code>	<code>awsSTSRegion</code>

AWS IAM Role-based Authorization

With role-based authorization, Redshift cluster temporarily assumes an IAM role when executing `COPY SQL`. You need to provide the role Amazon Resource Number (ARN) as a configuration value as follows: `gg.eventhandler.redshift.AwsIamRole`. For example: `gg.eventhandler.redshift.AwsIamRole=arn:aws:iam:<aws_account_id>:role/<role_name>`. The role needs to be authorized to read the respective S3 bucket. Ensure that the trust relationship of the role contains the AWS redshift service. Additionally, attach this role to the Redshift cluster before starting the Redshift cluster. For example, AWS IAM policy that can be used in the the trust relationship of the role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "redshift.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

If the role-based authorization is configured (`gg.eventhandler.redshift.AwsIamRole`), then it is given priority over key-based authorization.

10.2.5.11 Co-ordinated Apply Support

To enable co-ordinated apply for Redshift, ensure that the Redshift database's isolation level is set to SNAPSHOT. The Redshift SNAPSHOT ISOLATION option allows higher concurrency, where concurrent modifications to different rows in the same table can complete successfully.

SQL Query to Alter the Database's Isolation Level

```
ALTER DATABASE <sampledb> ISOLATION LEVEL SNAPSHOT;
```

10.2.5.12 Support for Mixed Case Identifiers

Oracle GoldenGate Redshift Event handler now supports mixed case names in the Replicat MAP statement. Mixed cases identifiers need to be enclosed inside double quotes as per the following example:

```
MAP QASOURCE.TCUSTMER, TARGET "QaTarget"."TCustmer";
```

10.2.5.13 AWS Secrets Manager Authentication

To securely manage and retrieve Redshift credentials, Oracle GoldenGate supports integration with AWS Secrets Manager. When using Secrets Manager for authentication, configure the following properties in your replicat properties file:

- `gg.eventhandler.redshift.secretsManagerSecretId`: Specify the id of the secret in AWS Secrets Manager containing your Redshift credentials.
- `gg.eventhandler.redshift.secretsManagerRegion`: Specify the AWS region where the secret is stored.

Note

- Do not configure the properties `gg.eventhandler.redshift.UserName` and `gg.eventhandler.redshift.Password` while using the AWS Secrets Manager. Otherwise, the set values of these properties will override the retrieved credentials from the secret when configuring the replicat.
- S3 credentials must be provided as environment variables under the names `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` respectively. Credentials set using the `gg.eventhandler.s3.accesskeyid`, `gg.eventhandler.s3.secretkey`, or `gg.eventhandler.s3.sessionToken` properties will not be recognized.

10.2.5.14 Restrictions on Database Object Names

- **Catalog, Schema, and Table Names:** Cannot contain period (`.`), asterisk (`*`), question mark (`?`), double quote (`"`), backslash (`\`), or control characters (ASCII 0-31).
- **Specific limitation for GGDA 21c and previous releases:** Supports only names that comply with the Avro specification for identifier names: must start with `[A-Z a-z_]`, followed by `[A-Za-z0-9_]`
- Additionally, all object names are also limited by the supported list of characters for the respective target.

10.2.6 Amazon S3

Learn how to use the S3 Event Handler, which provides the interface to Amazon S3 web services.

- [Overview](#)
- [Detailing Functionality](#)
- [Configuring the S3 Event Handler](#)
- [Enabling the Amazon S3 Multipart Upload](#)

10.2.6.1 Overview

Amazon S3 is object storage hosted in the Amazon cloud. The purpose of the S3 Event Handler is to load data files generated by the File Writer Handler into Amazon S3, see <https://aws.amazon.com/s3/>.

You can use any format that the File Writer Handler, see [Flat Files](#).

10.2.6.2 Detailing Functionality

The S3 Event Handler requires the Amazon Web Services (AWS) Java SDK to transfer files to S3 object storage. Oracle GoldenGate for Big Data does not include the AWS Java SDK. 1.x AWS Java SDK versions are no longer supported, it is recommended to use 2.28.11 or higher. You have to download and install the AWS Java SDK from:

<https://aws.amazon.com/sdk-for-java/>

Then you have to configure the `gg.classpath` variable to include the JAR files in the AWS Java SDK and are divided into two directories. Both directories must be in `gg.classpath`, for example:

```
gg.classpath=/usr/var/aws_sdk_2.28.11/*:/usr/var/aws_sdk_2.28.11/third-party/lib/
```

- [Resolving AWS Credentials](#)
- [About the AWS S3 Buckets](#)
- [Troubleshooting](#)

10.2.6.2.1 Resolving AWS Credentials

- [Amazon Web Services Simple Storage Service Client Authentication](#)
The S3 Event Handler is a client connection to the Amazon Web Services (AWS) Simple Storage Service (S3) cloud service. The AWS cloud must be able to successfully authenticate the AWS client in order in order to successfully interface with S3.

10.2.6.2.1.1 Amazon Web Services Simple Storage Service Client Authentication

The S3 Event Handler is a client connection to the Amazon Web Services (AWS) Simple Storage Service (S3) cloud service. The AWS cloud must be able to successfully authenticate the AWS client in order in order to successfully interface with S3.

The AWS client authentication has become increasingly complicated as more authentication options have been added to the S3 Event Handler. This topic explores the different use cases for AWS client authentication.

- [Explicit Configuration of the Client ID and Secret](#)
A client ID and secret are generally the required credentials for the S3 Event Handler to interact with Amazon S3. A client ID and secret are generated using the Amazon AWS website.
- [Use of the AWS Default Credentials Provider Chain](#)
If the `gg.eventhandler.name.accessKeyId` and `gg.eventhandler.name.secretKey` are unset, then credentials resolution reverts to the AWS default credentials provider chain. The AWS default credentials provider chain provides various ways by which the AWS credentials can be resolved.
- [AWS Federated Login](#)
The use case is when you have your on-premise system login integrated with AWS. This means that when you log into an on-premise machine, you are also logged into AWS.

10.2.6.2.1.1 Explicit Configuration of the Client ID and Secret

A client ID and secret are generally the required credentials for the S3 Event Handler to interact with Amazon S3. A client ID and secret are generated using the Amazon AWS website.

These credentials can be explicitly configured in the Java Adapter Properties file as follows:

```
gg.eventhandler.name.accessKeyId=  
gg.eventhandler.name.secretKey=
```

Furthermore, the Oracle Wallet functionality can be used to encrypt these credentials.

10.2.6.2.1.2 Use of the AWS Default Credentials Provider Chain

If the `gg.eventhandler.name.accessKeyId` and `gg.eventhandler.name.secretKey` are unset, then credentials resolution reverts to the AWS default credentials provider chain. The AWS default credentials provider chain provides various ways by which the AWS credentials can be resolved.

For more information about the default credential provider chain and order of operations for AWS credentials resolution, see [Working with AWS Credentials](#).

When Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) runs on an AWS Elastic Compute Cloud (EC2) instance, the general use case is to resolve the credentials from the EC2 metadata service. The AWS default credentials provider chain provides resolution of credentials from the EC2 metadata service as one of the options.

10.2.6.2.1.3 AWS Federated Login

The use case is when you have your on-premise system login integrated with AWS. This means that when you log into an on-premise machine, you are also logged into AWS.

In this use case:

- You may not want to generate client IDs and secrets. (Some users disable this feature in the AWS portal).
- The client AWS applications need to interact with the AWS Security Token Service (STS) to obtain an authentication token for programmatic calls made to S3.

This feature is enabled by setting the following: `gg.eventhandler.name.enableSTS=true`.

10.2.6.2.2 About the AWS S3 Buckets

AWS divides S3 storage into separate file systems called **buckets**. The S3 Event Handler can write to pre-created buckets. Alternatively, if the S3 bucket does not exist, the S3 Event Handler attempts to create the specified S3 bucket. AWS requires that S3 bucket names are

lowercase. Amazon S3 bucket names must be globally unique. If you attempt to create an S3 bucket that already exists in any Amazon account, it causes the S3 Event Handler to abend.

10.2.6.2.3 Troubleshooting

Connectivity Issues

If the S3 Event Handler is unable to connect to the S3 object storage when running on premise, it's likely your connectivity to the public internet is protected by a proxy server. Proxy servers act a gateway between the private network of a company and the public internet. Contact your network administrator to get the URLs of your proxy server.

Oracle GoldenGate can be used with a proxy server using the following parameters to enable the proxy server:

```
gg.handler.name.proxyServer=  
gg.handler.name.proxyPort=80  
gg.handler.name.proxyUsername=username  
gg.handler.name.proxyPassword=password
```

Sample configuration:

```
gg.eventhandler.s3.type=s3  
gg.eventhandler.s3.region=us-west-2  
gg.eventhandler.s3.proxyServer=www-proxy.us.oracle.com  
gg.eventhandler.s3.proxyPort=80  
gg.eventhandler.s3.proxyProtocol=HTTP  
gg.eventhandler.s3.bucketMappingTemplate=yourbucketname  
gg.eventhandler.s3.pathMappingTemplate=thepath  
gg.eventhandler.s3.finalizeAction=none
```

Duplicate records after Replicat Recovery

s3 replication uses File Writer Handler and s3 Handler in the replicat. Oracle GoldenGate prioritizes no data loss and guarantees no data loss in case of failures by at least once semantics in s3 (json, csv, delimitedtext, avro_orc, parquet) delivery. In cases where the replicat runs fine and normally shuts down, exactly once is supported. In case of failures (because of network failures), there are various reason that can lead into duplicates in recovery.

Two cases where duplicates can occur:

1. If data is written and a failure occurs between when the data is written, and when the checkpoint is moved. Then upon restart the replicat backs up to the previous checkpoint and data can unfortunately be replayed.
2. The rolling of the data files occurs based on customer configured triggers. Trigger can be file size, time, inactivity, or time of day. The rolling does not necessarily happen on a transaction commit boundary. The trigger causes writing to the current file to complete, the post-processing transformation, movement complete, and any state on that file is deleted. If a replicat abend occurs in between when the rolling is processed and when the checkpoint is moved, then upon restart, it can again replay those messages.

If you observe duplicate records in case of s3 replicat recovery, then it is an expected behavior. If you observe duplicates while replicat is running fine, then file a support ticket.

10.2.6.3 Configuring the S3 Event Handler

You can configure the S3 Event Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the S3 Event Handler, you must first configure the handler type by specifying `gg.eventhandler.name.type=s3` and the other S3 Event properties as follows:

Table 10-6 S3 Event Handler Configuration Properties

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.type</code>	Required	s3	None	Selects the S3 Event Handler for use with Replicat.
<code>gg.eventhandler.name.region</code>	Required	The AWS region name that is hosting your S3 instance.	None	Setting the legal AWS region name is required.
<code>gg.eventhandler.name.cannedACL</code>	Optional	Accepts one of the following values: - private - public-read - public-read-write - aws-exec-read - authenticated-read - bucket-owner-read - bucket-owner-full-control - log-deliver-write	None	Amazon S3 supports a set of predefined grants, known as canned Access Control Lists. Each canned ACL has a predefined set of grantees and permissions. For more information, see Managing access with ACLs
<code>gg.eventhandler.name.proxyServer</code>	Optional	The host name of your proxy server.	None	Sets the host name of your proxy server if connectivity to AWS is required use a proxy server.
<code>gg.eventhandler.name.proxyPort</code>	Optional	The port number of the proxy server.	None	Sets the port number of the proxy server if connectivity to AWS is required use a proxy server.

Table 10-6 (Cont.) S3 Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.proxyUserName</code>	Optional	The username of the proxy server.	None	Sets the user name of the proxy server if connectivity to AWS is required use a proxy server and the proxy server requires credentials.
<code>gg.eventhandler.name.proxyPassword</code>	Optional	The password of the proxy server.	None	Sets the password for the user name of the proxy server if connectivity to AWS is required use a proxy server and the proxy server requires credentials.
<code>gg.eventhandler.name.bucketMappingTemplate</code>	Required	A string with resolvable keywords and constants used to dynamically generate the path in the S3 bucket to write the file.	None	Use resolvable keywords and constants used to dynamically generate the S3 bucket name at runtime. The handler attempts to create the S3 bucket if it does not exist. AWS requires bucket names to be all lowercase. A bucket name with uppercase characters results in a runtime exception. See Template Keywords .
<code>gg.eventhandler.name.pathMappingTemplate</code>	Required	A string with resolvable keywords and constants used to dynamically generate the path in the S3 bucket to write the file.	None	Use keywords interlaced with constants to dynamically generate unique S3 path names at runtime. Typically, path names follow the format, <code>ogg/data/\${groupName}/\${fullyQualifiedTableName}</code> In S3, the convention is <i>not</i> to begin the path with the backslash (/) because it results in a root directory of "/". See Template Keywords .
<code>gg.eventhandler.name.fileNameMappingTemplate</code>	Optional	A string with resolvable keywords and constants used to dynamically generate the S3 file name at runtime.	None	Use resolvable keywords and constants used to dynamically generate the S3 data file name at runtime. If not set, the upstream file name is used. See Template Keywords .
<code>gg.eventhandler.name.finalizeAction</code>	Optional	none delete	None	Set to none to leave the S3 data file in place on the finalize action. Set to delete if you want to delete the S3 data file with the finalize action.

Table 10-6 (Cont.) S3 Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler. .name.eventHandler</code>	Optional	A unique string identifier cross referencing a child event handler.	No event handler configured.	Sets the event handler that is invoked on the file roll event. Event handlers can do file roll event actions like loading files to S3, converting to Parquet or ORC format, or loading files to HDFS.
<code>gg.eventhandler. .name.url</code>	Optional	A legal URL to connect to cloud storage.	None	Used to set the URL endpoint for S3. It can be used for connecting to Amazon S3 private endpoints or S3 API compatible services like Dell ECS.
<code>gg.eventhandler. .name.proxyProtocol</code>	Optional	HTTP HTTPS	HTTP	Sets the proxy protocol connection to the proxy server for additional level of security. The client first performs an SSL handshake with the proxy server, and then an SSL handshake with Amazon AWS. This feature was added into the Amazon SDK in version 1.11.396 so you must use at least that version to use this property.
<code>gg.eventhandler. .name.SSEAlgorithm</code>	Optional	AES256 aws:kms	Empty	Set only if you are enabling S3 server side encryption. Use the parameters to set the algorithm for server side encryption in S3.
<code>gg.eventhandler. .name.AWSKmsKeyId</code>	Optional	A legal AWS key management system server side management key or the alias that represents that key.	Empty	Set only if you are enabling S3 server side encryption and the S3 algorithm is <code>aws:kms</code> . This is either the encryption key or the encryption alias that you set in the AWS Identity and Access Management web page. Aliases are prepended with <code>alias/</code> .
<code>gg.eventhandler. .name.enableSTS</code>	Optional	true false	false	Set to true, to enable the S3 Event Handler to access S3 credentials from the AWS Security Token Service. The AWS Security Token Service must be enabled if you set this property to true.
<code>gg.eventhandler. .name.STSAsumeRole</code>	Optional	AWS user and role in the following format: {user arn}:role /{role name}	None	Set configuration if you want to assume a different user/role. Only valid with STS enabled.

Table 10-6 (Cont.) S3 Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.STSAssumeRoleSessionName</code>	Optional	Any string.	AssumeRoleSession1	The assumed role requires a session name for session logging. However this can be any value. Only valid if both <code>gg.eventhandler.name.enableSTS=true</code> and <code>gg.eventhandler.name.STSAssumeRole</code> are configured.
<code>gg.eventhandler.name.STSRegion</code>	Optional	Any legal AWS region specifier.	The region is obtained from the <code>gg.eventhandler.name.region</code> property.	Use to resolve the region for the STS call. It's only valid if the <code>gg.eventhandler.name.enableSTS</code> property is set to true. You can set a different AWS region for resolving credentials from STS than the configured S3 region.
<code>gg.eventhandler.name.enableBucketAdmin</code>	Optional	true false	true	Set to false to disable checking if S3 buckets exist and automatic creation of buckets, if they do not exist. This feature requires S3 admin privileges on S3 buckets which some customers do not wish to grant.
<code>gg.eventhandler.name.accessKeyId</code>	Optional	A valid AWS access key.	None	Set this parameter to explicitly set the access key for AWS. This parameter has no effect if <code>gg.eventhandler.name.enableSTS</code> is set to true. If this property is not set, then the credentials resolution falls back to the AWS default credentials provider chain.
<code>gg.eventhandler.name.secretKey</code>	Optional	A valid AWS secret key.	None	Set this parameter to explicitly set the secret key for AWS. This parameter has no effect if <code>gg.eventhandler.name.enableSTS</code> is set to true. If this property is not set, then credentials resolution falls back to the AWS default credentials provider chain.
<code>gg.eventhandler.s3.enableAccelerateMode</code>	Optional	true false	false	Enable/Disable Amazon S3 Transfer Acceleration to transfer files quickly and securely over long distances between your client and an S3 bucket.
<code>gg.eventhandler.s3.purgeTargetPathOnStartup</code>	Optional	true false	false	If the value is set to true, all the existing contents inside the S3 target, will be deleted before replicating the data.
<code>gg.eventhandler.name.multipartEnabled</code>	Optional	true false	false	If set to true, files bigger than the configured threshold will be uploaded to S3 using the multipart upload feature.

Table 10-6 (Cont.) S3 Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.apiCallBufferSizeInBytes</code>	Optional	A long type value	32 MiB (33554432 bytes)	This is a multipart upload configuration property, so <code>gg.eventhandler.s3.multipartEnabled</code> must be set to <code>true</code> for this to be used. This configures the maximum amount of memory, in bytes, the SDK will use to buffer content of requests in memory. Increasing this value may lead to better performance at the cost of using more memory.
<code>gg.eventhandler.name.minimumPartSizeInBytes</code>	Optional	A long type value	8 MiB (8388608 bytes)	This is a multipart upload configuration property. The <code>gg.eventhandler.s3.multipartEnabled</code> must be set to <code>true</code> to use this property. This configures the minimum part size, in bytes, to be used in each individual part requests.
<code>gg.eventhandler.name.thresholdInBytes</code>	Optional	A long type value	8 MiB (8388608 bytes)	This is a multipart upload configuration property. The <code>gg.eventhandler.s3.multipartEnabled</code> must be set to <code>true</code> to use this property. This configures the size threshold, in bytes, for when to use the multipart upload. It specifies that the uploads over the configured size will automatically use the multipart upload strategy, while uploads smaller than this threshold value will use a single connection to upload the complete object.
<code>gg.eventhandler.name.maxConcurrency</code>	Required	An integer type value	250	The maximum number of allowed concurrent requests.
<code>gg.eventhandler.name.connectionAcquisitionTimeoutInSeconds</code>	required	An integer type value	600	The amount of time, in seconds, to wait when acquiring a connection from the pool before giving up and timing out.

10.2.6.4 Enabling the Amazon S3 Multipart Upload

Enabling the Amazon S3 Multipart Upload feature allows uploading the large files to Amazon S3 by splitting them into small parts that can be uploaded independently and in parallel. This feature solves the already known Amazon S3 limitations, such as the single upload of 5GB+ size objects. Once all the file parts are successfully uploaded, a request is made to Amazon S3 to assemble them into a single object.

Note

The Amazon S3 Multipart Upload feature can only be used in objects larger than the configured threshold, objects smaller than the threshold can still be uploaded as a complete single part.

The Amazon S3 Multipart Upload feature is disabled by default in the Amazon S3 Oracle GoldenGate handler. To use this feature you need to enable the following property :

```
gg.eventhandler.s3.multipartEnabled=true
```

Once enabled, you can use the following properties to configure the Amazon S3 Multipart Upload settings:

Table 10-7 Amazon S3 Multipart Upload Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler.s3.apiCallBuffer erSizeInBytes	Optional	A long type number value	32 MiB (335544 32 bytes)	It configures the maximum amount of memory, in bytes, the SDK uses to buffer content of requests in memory. Increasing this value may lead to better performance at the cost of using more memory.
gg.eventhandler.s3.minimumPart SizeInBytes	Optional	A long type number value	8 MiB (838860 8 bytes)	It configures the minimum part size, in bytes, to be used in each individual part requests.
gg.eventhandler.s3.thresholdIn Bytes	Optional	A long type number value	8 MiB (838860 8 bytes)	It configures the size threshold, in bytes, for when to use the multipart upload. Uploads over this size automatically uses the multipart upload, while uploads smaller than this threshold uses a single connection to upload the whole object.

Note

These configuration properties are only required, if they are set, if not, the SDK uses the default values for each property, that is, 32 Mib, 8 Mib, and 8 Mib respectively.

10.2.7 Apache Cassandra

The Cassandra Handler provides the interface to Apache Cassandra databases.

This chapter describes how to use the Cassandra Handler.

- [Overview](#)
- [Detailing the Functionality](#)
- [Setting Up and Running the Cassandra Handler](#)
- [About Automated DDL Handling](#)
The Cassandra Handler performs the table check and reconciliation process the first time an operation for a source table is encountered. Additionally, a DDL event or a metadata change event causes the table definition in the Cassandra Handler to be marked as not suitable.
- [Performance Considerations](#)
- [Additional Considerations](#)
- [Troubleshooting](#)

- [Cassandra Handler Client Dependencies](#)
What are the dependencies for the Cassandra Handler to connect to Apache Cassandra databases?

10.2.7.1 Overview

Apache Cassandra is a NoSQL Database Management System designed to store large amounts of data. A Cassandra cluster configuration provides horizontal scaling and replication of data across multiple machines. It can provide high availability and eliminate a single point of failure by replicating data to multiple nodes within a Cassandra cluster. Apache Cassandra is open source and designed to run on low-cost commodity hardware.

Cassandra relaxes the axioms of a traditional relational database management systems (RDBMS) regarding atomicity, consistency, isolation, and durability. When considering implementing Cassandra, it is important to understand its differences from a traditional RDBMS and how those differences affect your specific use case.

Cassandra provides eventual consistency. Under the eventual consistency model, accessing the state of data for a specific row eventually returns the latest state of the data for that row as defined by the most recent change. However, there may be a latency period between the creation and modification of the state of a row and what is returned when the state of that row is queried. The benefit of eventual consistency is that the latency period is predicted based on your Cassandra configuration and the level of work load that your Cassandra cluster is currently under, see <http://cassandra.apache.org/>.

The Cassandra Handler provides some control over consistency with the configuration of the `gg.handler.name.consistencyLevel` property in the Java Adapter properties file.

10.2.7.2 Detailing the Functionality

- [About the Cassandra Data Types](#)
- [About Catalog, Schema, Table, and Column Name Mapping](#)
Traditional RDBMSs separate structured data into tables. Related tables are included in higher-level collections called databases. Cassandra contains both of these concepts. Tables in an RDBMS are also tables in Cassandra, while database schemas in an RDBMS are keyspaces in Cassandra.
- [About DDL Functionality](#)
- [How Operations are Processed](#)
- [About Compressed Updates vs. Full Image Updates](#)
- [About Primary Key Updates](#)

10.2.7.2.1 About the Cassandra Data Types

Cassandra provides a number of column data types and most of these data types are supported by the Cassandra Handler.

Supported Cassandra Data Types

ASCII
BIGINT
BLOB
BOOLEAN
DATE

DECIMAL
DOUBLE
DURATION
FLOAT
INET
INT
SMALLINT
TEXT
TIME
TIMESTAMP
TIMEUUID
TINYINT
UUID
VARCHAR
VARINT

Unsupported Cassandra Data Types

COUNTER
MAP
SET
LIST
UDT (user defined type)
TUPLE
CUSTOM_TYPE

Supported Database Operations

INSERT
UPDATE (captured as INSERT)
DELETE

The Cassandra commit log files do *not* record any before images for the UPDATE or DELETE operations. So the captured operations never have a before image section. Oracle GoldenGate features that rely on before image records, such as Conflict Detection and Resolution, are not available.

Unsupported Database Operations

TRUNCATE
DDL (CREATE, ALTER, DROP)

The data type of the column value in the source trail file must be converted to the corresponding Java type representing the Cassandra column type in the Cassandra Handler. This data conversion introduces the risk of a runtime conversion error. A poorly mapped field (such as varchar as the source containing alpha numeric data to a Cassandra int) may cause a runtime error and cause the Cassandra Handler to abend. You can view the Cassandra Java type mappings at:

[DataStax Documentation](#)

It is possible that the data may require specialized processing to get converted to the corresponding Java type for intake into Cassandra. If this is the case, you have two options:

- Try to use the general regular expression search and replace functionality to format the source column value data in a way that can be converted into the Java data type for use in Cassandra.
- Or
- Implement or extend the default data type conversion logic to override it with custom logic for your use case. Contact Oracle Support for guidance.

10.2.7.2.2 About Catalog, Schema, Table, and Column Name Mapping

Traditional RDBMSs separate structured data into tables. Related tables are included in higher-level collections called databases. Cassandra contains both of these concepts. Tables in an RDBMS are also tables in Cassandra, while database schemas in an RDBMS are keyspaces in Cassandra.

It is important to understand how data maps from the metadata definition in the source trail file are mapped to the corresponding keyspace and table in Cassandra. Source tables are generally either two-part names defined as `schema.table`, or three-part names defined as `catalog.schema.table`.

The following table explains how catalog, schema, and table names map into Cassandra. Unless you use special syntax, Cassandra converts all keyspace, table names, and column names to lower case.

Table Name in Source Trail File	Cassandra Keyspace Name	Cassandra Table Name
QASOURCE.TCUSTMER	qasource	tcustmer
dbo.mytable	dbo	mytable
GG.QASOURCE.TCUSTORD	gg_qasource	tcustord

10.2.7.2.3 About DDL Functionality

- [About the Keyspaces](#)
- [About the Tables](#)
- [Adding Column Functionality](#)
- [Dropping Column Functionality](#)

10.2.7.2.3.1 About the Keyspaces

The Cassandra Handler does *not* automatically create keyspaces in Cassandra. Keyspaces in Cassandra define a replication factor, the replication strategy, and topology. The Cassandra Handler does not have enough information to create the keyspaces, so you must manually create them.

You can create keyspaces in Cassandra by using the `CREATE KEYSPACE` command from the Cassandra shell.

10.2.7.2.3.2 About the Tables

The Cassandra Handler can automatically create tables in Cassandra if you configure it to do so. The source table definition may be a poor source of information to create tables in Cassandra. Primary keys in Cassandra are divided into:

- **Partitioning keys** that define how data for a table is separated into partitions in Cassandra.
- **Clustering keys** that define the order of items within a partition.

In the default mapping for automated table creation, the first primary key is the partition key, and any additional primary keys are mapped as clustering keys.

Automated table creation by the Cassandra Handler may be fine for proof of concept, but it may result in data definitions that do not scale well. When the Cassandra Handler creates tables with poorly constructed primary keys, the performance of ingest and retrieval may decrease as the volume of data stored in Cassandra increases. Oracle recommends that you analyze the metadata of your replicated tables, then manually create corresponding tables in Cassandra that are properly partitioned and clustered for higher scalability.

Primary key definitions for tables in Cassandra are immutable after they are created. Changing a Cassandra table primary key definition requires the following manual steps:

1. Create a staging table.
2. Populate the data in the staging table from original table.
3. Drop the original table.
4. Re-create the original table with the modified primary key definitions.
5. Populate the data in the original table from the staging table.
6. Drop the staging table.

10.2.7.2.3.3 Adding Column Functionality

You can configure the Cassandra Handler to add columns that exist in the source trail file table definition but are missing in the Cassandra table definition. The Cassandra Handler can accommodate metadata change events of this kind. A reconciliation process reconciles the source table definition to the Cassandra table definition. When the Cassandra Handler is configured to add columns, any columns found in the source table definition that do not exist in the Cassandra table definition are added. The reconciliation process for a table occurs after application startup the first time an operation for the table is encountered. The reconciliation process reoccurs after a metadata change event on a source table, when the first operation for the source table is encountered after the change event.

10.2.7.2.3.4 Dropping Column Functionality

You can configure the Cassandra Handler to drop columns that do not exist in the source trail file definition but exist in the Cassandra table definition. The Cassandra Handler can accommodate metadata change events of this kind. A reconciliation process reconciles the source table definition to the Cassandra table definition. When the Cassandra Handler is configured to drop, columns any columns found in the Cassandra table definition that are not in the source table definition are dropped.

Caution

Dropping a column permanently removes data from a Cassandra table. Carefully consider your use case before you configure this mode.

Note

Primary key columns cannot be dropped. Attempting to do so results in an abend.

Note

Column name changes are not well-handled because there is no DDL is processed. When a column name changes in the source database, the Cassandra Handler interprets it as dropping an existing column and adding a new column.

10.2.7.2.4 How Operations are Processed

The Cassandra Handler pushes operations to Cassandra using either the asynchronous or synchronous API. In asynchronous mode, operations are flushed at transaction commit (grouped transaction commit using GROUPTRANSOPS) to ensure write durability. The Cassandra Handler does not interface with Cassandra in a transactional way.

Supported Database Operations

INSERT
UPDATE (captured as INSERT)
DELETE

The Cassandra commit log files do *not* record any before images for the UPDATE or DELETE operations. So the captured operations never have a before image section. Oracle GoldenGate features that rely on before image records, such as Conflict Detection and Resolution, are not available.

Unsupported Database Operations

TRUNCATE
DDL (CREATE, ALTER, DROP)

Insert, update, and delete operations are processed differently in Cassandra than a traditional RDBMS. The following explains how insert, update, and delete operations are interpreted by Cassandra:

- Inserts: If the row does not exist in Cassandra, then an insert operation is processed as an insert. If the row already exists in Cassandra, then an insert operation is processed as an update.
- Updates: If a row does not exist in Cassandra, then an update operation is processed as an insert. If the row already exists in Cassandra, then an update operation is processed as insert.
- Delete: If the row does not exist in Cassandra, then a delete operation has no effect. If the row exists in Cassandra, then a delete operation is processed as a delete.

The state of the data in Cassandra is idempotent. You can replay the source trail files or replay sections of the trail files. The state of the Cassandra database must be the same regardless of the number of times that the trail data is written into Cassandra.

10.2.7.2.5 About Compressed Updates vs. Full Image Updates

Oracle GoldenGate allows you to control the data that is propagated to the source trail file in the event of an update. The data for an update in the source trail file is either a compressed or a full image of the update, and the column information is provided as follows:

Compressed

For the primary keys and the columns for which the value changed. Data for columns that have not changed is not provided in the trail file.

Full Image

For all columns, including primary keys, columns for which the value has changed, and columns for which the value has not changed.

The amount of information about an update is important to the Cassandra Handler. If the source trail file contains full images of the change data, then the Cassandra Handler can use prepared statements to perform row updates in Cassandra. Full images also allow the Cassandra Handler to perform primary key updates for a row in Cassandra. In Cassandra, primary keys are immutable, so an update that changes a primary key must be treated as a delete and an insert. Conversely, when compressed updates are used, prepared statements cannot be used for Cassandra row updates. Simple statements identifying the changing values and primary keys must be dynamically created and then executed. With compressed updates, primary key updates are not possible and as a result, the Cassandra Handler will abend.

You must set the control properties `gg.handler.name.compressedUpdates` and `gg.handler.name.compressedUpdatesfor` so that the handler expects either compressed or full image updates.

The default value, `true`, sets the Cassandra Handler to expect compressed updates. Prepared statements are not be used for updates, and primary key updates cause the handler to abend.

When the value is `false`, prepared statements are used for updates and primary key updates can be processed. A source trail file that does not contain full image data can lead to corrupted data columns, which are considered null. As a result, the null value is pushed to Cassandra. If you are not sure about whether the source trail files contains compressed or full image data, set `gg.handler.name.compressedUpdates` to `true`.

CLOB and BLOB data types do not propagate LOB data in updates unless the LOB column value changed. Therefore, if the source tables contain LOB data, set `gg.handler.name.compressedUpdates` to `true`.

10.2.7.2.6 About Primary Key Updates

Primary key values for a row in Cassandra are immutable. An update operation that changes any primary key value for a Cassandra row must be treated as a delete and insert. The Cassandra Handler can process update operations that result in the change of a primary key in Cassandra only as a delete and insert. To successfully process this operation, the source trail file *must* contain the complete before and after change data images for all columns. The `gg.handler.name.compressed` configuration property of the Cassandra Handler must be set to `false` for primary key updates to be successfully processed.

10.2.7.3 Setting Up and Running the Cassandra Handler

Instructions for configuring the Cassandra Handler components and running the handler are described in the following sections.

Before you run the Cassandra Handler, you must install the Datastax Driver for Cassandra and set the `gg.classpath` configuration property.

Get the Driver Libraries

The Cassandra Handler has been updated to use the newer 4.x versions of the Datastax Java Driver or 2.x versions of the Datastax Enterprise Java Driver. The Datastax Java Driver for Cassandra does not ship with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA). For more information, see

[Datastax Java Driver for Apache Cassandra](#).

You can use the Dependency Downloader scripts to download the Datastax Java Driver and its associated dependencies.

Set the Classpath

You must configure the `gg.classpath` configuration property in the Java Adapter properties file to specify the JARs for the Datastax Java Driver for Cassandra. Ensure that this JAR is first in the list.

```
gg.classpath=/path/to/4.x/cassandra-java-driver/*
```

- [Understanding the Cassandra Handler Configuration](#)
- [Review a Sample Configuration](#)
- [Configuring Security](#)

10.2.7.3.1 Understanding the Cassandra Handler Configuration

The following are the configurable values for the Cassandra Handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the Cassandra Handler, you must first configure the handler type by specifying `gg.handler.name.type=cassandra` and the other Cassandra properties as follows:

Table 10-8 Cassandra Handler Configuration Properties

Properties	Require d/ Optiona l	Legal Values	Default	Explanation
<code>gg.handlerlist</code>	Require d	Any string	None	Provides a name for the Cassandra Handler. The Cassandra Handler name then becomes part of the property names listed in this table.
<code>gg.handler.name.type=cassandra</code>	Require d	cassandr a	None	Selects the Cassandra Handler for streaming change data capture into name.
<code>gg.handler.name.mode</code>	Optional	op tx	op	The default is recommended. In op mode, operations are processed as received. In tx mode, operations are cached and processed at transaction commit. The txmode is slower and creates a larger memory footprint.
<code>gg.handler.name.conta ctPoints=</code>	Optional	A comma separated list of host names that the Cassandr a Handler will connect to.	localhost	A comma-separated list of the Cassandra host machines for the driver to establish an initial connection to the Cassandra cluster. This configuration property does <i>not</i> need to include all the machines enlisted in the Cassandra cluster. By connecting to a single machine, the driver can learn about other machines in the Cassandra cluster and establish connections to those machines as required.

Table 10-8 (Cont.) Cassandra Handler Configuration Properties

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.handler.name.username</code>	Optional	A legal username string.	None	A user name for the connection to name. Required if Cassandra is configured to require credentials.
<code>gg.handler.name.password</code>	Optional	A legal password string.	None	A password for the connection to name. Required if Cassandra is configured to require credentials.
<code>gg.handler.name.compressedUpdates</code>	Optional	true false	true	Sets the Cassandra Handler whether to expect full image updates from the source trail file. A value of <code>true</code> means that updates in the source trail file only contain column data for the primary keys and for columns that changed. The Cassandra Handler executes updates as simple statements updating only the columns that changed. A value of <code>false</code> means that updates in the source trail file contain column data for primary keys and all columns regardless of whether the column value has changed. The Cassandra Handler is able to use prepared statements for updates, which can provide better performance for streaming data to name.
<code>gg.handler.name.ddlHandling</code>	Optional	CREATE ADD DROP in any combination with values delimited by a comma	None	Configures the Cassandra Handler for the DDL functionality to provide. Options include CREATE, ADD, and DROP. These options can be set in any combination delimited by commas. When CREATE is enabled, the Cassandra Handler creates tables in Cassandra if a corresponding table does not exist. When ADD is enabled, the Cassandra Handler adds columns that exist in the source table definition that do <i>not</i> exist in the corresponding Cassandra table definition. When DROP is enabled, the handler drops columns that exist in the Cassandra table definition that do <i>not</i> exist in the corresponding source table definition.

Table 10-8 (Cont.) Cassandra Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.cassandraMode</code>	Optional	async sync	async	Sets the interaction between the Cassandra Handler and name. Set to async for asynchronous interaction. Operations are sent to Cassandra asynchronously and then flushed at transaction commit to ensure durability. Asynchronous provides better performance. Set to sync for synchronous interaction. Operations are sent to Cassandra synchronously.
<code>gg.handler.name.consistencyLevel</code>	Optional	ALL ANY EACH_QUORUM LOCAL_ONE LOCAL_QUORUM ONE QUORUM THREE TWO	The Cassandra default.	Sets the consistency level for operations with name. It configures the criteria that must be met for storage on the Cassandra cluster when an operation is executed. Lower levels of consistency may provide better performance, while higher levels of consistency are safer.
<code>gg.handler.name.port</code>	Optional	Integer	9042	Set to configure the port number that the Cassandra Handler attempts to connect to Cassandra server instances. You can override the default in the Cassandra YAML files.
<code>gg.handler.name.batchType</code>	Optional	String	unlogged	Sets the type for Cassandra batch processing. - unlogged - Does not use Cassandra's distributed batch log. - logged - Cassandra first writes to its distributed batch log to ensure atomicity of the batch. - counter - Use if counter types are updated in the batch.

Table 10-8 (Cont.) Cassandra Handler Configuration Properties

Properties	Require d/ Optiona l	Legal Values	Default	Explanation
<code>gg.handler.name.abendOnUnmappedColumns</code>	Optional	Boolean	true	Only applicable when <code>gg.handler.name.ddlHandling</code> is not configured with ADD. When set to true, the replicat process will abend if a column exists in the source table, but does not exist in the target Cassandra table. When set to false, the replicat process will not abend if a column exists in the source table, but does not exist in the target Cassandra table. Instead, that column will not be replicated.
<code>gg.handler.name.DatastaxJSSEConfigPath</code>	Optional	String	None	Set the path and file name of a properties file containing the Cassandra driver configuration. Use when the Cassandra driver configuration needs to be configured for non-default values and potentially SSL connectivity. For more information, see Cassandra Driver Configuration Documentation . You need to follow the syntax of the configuration file for the driver version you are using. The suffix of the Cassandra driver configuration file must be <code>.conf</code> .
<code>gg.handler.name.datacenter</code>	Optional	The datacenter name	datacenter 1	Set the datacenter name. If the datacenter name does not match the configured name on the server, then it will not connect to the database.

10.2.7.3.2 Review a Sample Configuration

The following is a sample configuration for the Cassandra Handler from the Java Adapter properties file:

```
gg.handlerlist=cassandra

#The handler properties
gg.handler.cassandra.type=cassandra
gg.handler.cassandra.mode=op
gg.handler.cassandra.contactPoints=localhost
gg.handler.cassandra.ddlHandling=CREATE,ADD,DROP
gg.handler.cassandra.compressedUpdates=true
gg.handler.cassandra.cassandraMode=async
gg.handler.cassandra.consistencyLevel=ONE
```

10.2.7.3.3 Configuring Security

The Cassandra Handler connection to the Cassandra Cluster can be secured using user name and password credentials. These are set using the following configuration properties:

```
gg.handler.name.username
gg.handler.name.password
```

Note

Security Recommendation: Oracle recommends storing credentials in Oracle GoldenGate Credential Store, Oracle Cloud Vault, or another enterprise credential management solution instead of storing plaintext passwords in configuration files.

To configure SSL, the recommendation is to configure the SSL properties via the Datastax Java Driver configuration file and point to the configuration file via the `gg.handler.name.DatastaxJSSEConfigPath` property. See <https://docs.datastax.com/en/developer/java-driver/4.14/manual/core/ssl/> for the SSL settings instructions.

Sample configuration file is as follows. Uncomment the relevant parameters and change to your required values.

```
datastax-java-driver {
  advanced.ssl-engine-factory {
    class = DefaultSslEngineFactory

    # This property is optional. If it is not present, the driver won't explicitly
    enable cipher
    # suites on the engine, which according to the JDK documentations results in "a
    minimum quality
    # of service".
    // cipher-suites = [ "TLS_RSA_WITH_AES_128_CBC_SHA", "TLS_RSA_WITH_AES_256_CBC_SHA" ]

    # Whether or not to require validation that the hostname of the server certificate's
    common
    # name matches the hostname of the server being connected to. If not set, defaults
    to true.
    // hostname-validation = true

    # The locations and passwords used to access truststore and keystore contents.
    # These properties are optional. If either truststore-path or keystore-path are
    specified,
    # the driver builds an SSLContext from these files. If neither option is specified,
    the
    # default SSLContext is used, which is based on system property configuration.
    // truststore-path = /path/to/client.truststore
    // truststore-password = password123
    // keystore-path = /path/to/client.keystore
    // keystore-password = password123
  }
}
```

10.2.7.4 About Automated DDL Handling

The Cassandra Handler performs the table check and reconciliation process the first time an operation for a source table is encountered. Additionally, a DDL event or a metadata change event causes the table definition in the Cassandra Handler to be marked as not suitable.

Therefore, the next time an operation for the table is encountered, the handler repeats the table check, and reconciliation process as described in this topic.

- [About the Table Check and Reconciliation Process](#)
- [Capturing New Change Data](#)

10.2.7.4.1 About the Table Check and Reconciliation Process

The Cassandra Handler first interrogates the target Cassandra database to determine whether the target Cassandra keyspace exists. If the target Cassandra keyspace does not exist, then the Cassandra Handler abends. Keyspaces must be created by the user. The log file must contain the error of the exact keyspace name that the Cassandra Handler is expecting.

Next, the Cassandra Handler interrogates the target Cassandra database for the table definition. If the table does not exist, the Cassandra Handler either creates a table if `gg.handler.name.ddlHandling` includes the CREATE option or abends the process. A message is logged that shows you the table that does not exist in Cassandra.

If the table exists in Cassandra, then the Cassandra Handler reconciles the table definition from the source trail file and the table definition in Cassandra. This reconciliation process searches for columns that exist in the source table definition and not in the corresponding Cassandra table definition. If it locates columns fitting this criteria and the `gg.handler.name.ddlHandling` property includes ADD, then the Cassandra Handler adds the columns to the target table in Cassandra. Otherwise, it ignores these columns.

Next, the Cassandra Handler searches for columns that exist in the target Cassandra table but do not exist in the source table definition. If it locates columns that fit this criteria and the `gg.handler.name.ddlHandling` property includes DROP, then the Cassandra Handler removes these columns from the target table in Cassandra. Otherwise those columns are ignored.

Finally, the prepared statements are built.

10.2.7.4.2 Capturing New Change Data

You can capture all of the new change data into your Cassandra database, including the DDL changes in the trail, for the target apply. Following is the acceptance criteria:

- AC1: Support Cassandra as a bulk extract
- AC2: Support Cassandra as a CDC source
- AC4: All Cassandra supported data types are supported
- AC5: Should be able to write into different tables based on any filter conditions, like Updates to Update tables or based on primary keys
- AC7: Support Parallel processing with multiple threads
- AC8: Support Filtering based on keywords
- AC9: Support for Metadata provider
- AC10: Support for DDL handling on sources and target
- AC11: Support for target creation and updating of metadata.
- AC12: Support for error handling and extensive logging
- AC13: Support for Conflict Detection and Resolution
- AC14: Performance should be on par or better than HBase

10.2.7.5 Performance Considerations

Configuring the Cassandra Handler for async mode provides better performance than sync mode. Set Replicat property GROUPTRANSOPS must be set to the default value of 1000.

Setting the consistency level directly affects performance. The higher the consistency level, the more work must occur on the Cassandra cluster before the transmission of a given operation can be considered complete. Select the minimum consistency level that still satisfies the requirements of your use case.

The Cassandra Handler can work in either operation (op) or transaction (tx) mode. For the best performance operation mode is recommended:

```
gg.handler.name.mode=op
```

10.2.7.6 Additional Considerations

- Cassandra database requires at least one primary key. The value of any primary key cannot be null. Automated table creation fails for source tables that do not have a primary key.
- When `gg.handler.name.compressedUpdates=false` is set, the Cassandra Handler expects to update full before and after images of the data.

Note

Using this property setting with a source trail file with partial image updates results in null values being updated for columns for which the data is missing. This configuration is incorrect and update operations pollute the target data with null values in columns that did not change.

- The Cassandra Handler does *not* process DDL from the source database, even if the source database provides DDL. Instead, it reconciles between the source table definition and the target Cassandra table definition. A DDL statement executed at the source database that changes a column name appears to the Cassandra Handler as if a column is dropped from the source table and a new column is added. This behavior depends on how the `gg.handler.name.ddlHandling` property is configured.

<code>gg.handler.name.ddlHandling</code> Configuration	Behavior
Not configured for ADD or DROP	Old column name and data maintained in Cassandra. New column is not created in Cassandra, so no data is replicated for the new column name from the DDL change forward.
Configured for ADD only	Old column name and data maintained in Cassandra. New column is created in Cassandra and data replicated for the new column name from the DDL change forward. Column mismatch between the data is located before and after the DDL change.
Configured for DROP only	Old column name and data dropped in Cassandra. New column is not created in Cassandra, so no data replicated for the new column name.

<code>gg.handler.name.ddlHandling</code> Configuration	Behavior
Configured for ADD and DROP	Old column name and data dropped in Cassandra. New column is created in Cassandra, and data is replicated for the new column name from the DDL change forward.

10.2.7.7 Troubleshooting

This section contains information to help you troubleshoot various issues.

- [Java Classpath](#)
- [Write Timeout Exception](#)
- [Datastax Driver Error](#)

10.2.7.7.1 Java Classpath

When the classpath that is intended to include the required client libraries, a `ClassNotFoundException` exception appears in the log file. To troubleshoot, set the Java Adapter logging to `DEBUG`, and then run the process again. At the debug level, the log contains data about the JARs that were added to the classpath from the `gg.classpath` configuration variable. The `gg.classpath` variable selects the asterisk (*) wildcard character to select all JARs in a configured directory. For example, `/usr/cassandra/cassandra-java-driver4.9.0/*:/usr/cassandra/cassandra-java-driver-4.9.0/lib/*`.

For more information about setting the classpath, see [Setting Up and Running the Cassandra Handler](#) and [Cassandra Handler Client Dependencies](#).

10.2.7.7.2 Write Timeout Exception

When running the Cassandra handler, you may experience a `com.datastax.driver.core.exceptions.WriteTimeoutException` exception that causes the Replicat process to abend. It is likely to occur under some or all of the following conditions:

- The Cassandra Handler processes large numbers of operations, putting the Cassandra cluster under a significant processing load.
- `GROUPTRANSOPS` is configured higher than the value of 1000 default.
- The Cassandra Handler is configured in asynchronous mode.
- The Cassandra Handler is configured with a consistency level higher than `ONE`.

When this problem occurs, the Cassandra Handler is streaming data faster than the Cassandra cluster can process it. The write latency in the Cassandra cluster finally exceeds the write request timeout period, which in turn results in the exception.

The following are potential solutions:

- Increase the write request timeout period. This is controlled with the `write_request_timeout_in_ms` property in Cassandra and is located in the `cassandra.yaml` file in the `cassandra_install/conf` directory. The default is 2000 (2 seconds). You can increase this value to move past the error, and then restart the Cassandra node or nodes for the change to take effect.
- Decrease the `GROUPTRANSOPS` configuration value of the Replicat process. Typically, decreasing the `GROUPTRANSOPS` configuration decreases the size of transactions processed and reduces the likelihood that the Cassandra Handler can overtax the Cassandra cluster.

- Reduce the consistency level of the Cassandra Handler. This in turn reduces the amount of work the Cassandra cluster has to complete for an operation to be considered as written.

10.2.7.7.3 Datastax Driver Error

The Cassandra Handler has been changed to use the 4.x version of the Datastax Java Driver. ClassNotFound exceptions can occur under either of the following conditions:

- The `gg.classpath` configuration is set to point at the old 3.x version of the Java Driver.
- The `gg.classpath` has not been configured to include the 4.x version of the Java Driver.

10.2.7.8 Cassandra Handler Client Dependencies

What are the dependencies for the Cassandra Handler to connect to Apache Cassandra databases?

The following Maven dependencies are required for the Cassandra Handler:

Artifact: `java-driver-core`

GroupId: `com.datastax.oss`

ArtifactId: `java-driver-core`

Version: `4.x`

Artifact: `java-driver-query-builder`

GroupId: `com.datastax.oss`

Artifact ID: `java-driver-query-builder`

Version: `4.x`

- [Cassandra Datastax Java Driver 4.12.0](#)
- [Cassandra Datastax Java Driver 4.9.0](#)

10.2.7.8.1 Cassandra Datastax Java Driver 4.12.0

asm-9.1.jar
asm-analysis-9.1.jar
asm-commons-9.1.jar
asm-tree-9.1.jar
asm-util-9.1.jar
config-1.4.1.jar
esri-geometry-api-1.2.1.jar
HdrHistogram-2.1.12.jar
jackson-annotations-2.12.2.jar
jackson-core-2.12.2.jar
jackson-core-asl-1.9.12.jar
jackson-databind-2.12.2.jar
java-driver-core-4.12.0.jar
java-driver-query-builder-4.12.0.jar
java-driver-shaded-guava-25.1-jre-graal-sub-1.jar
jcip-annotations-1.0-1.jar
jffi-1.3.1.jar
jffi-1.3.1-native.jar
jnr-a64asm-1.0.0.jar
jnr-constants-0.10.1.jar

jnr-ffi-2.2.2.jar
jnr-posix-3.1.5.jar
jnr-x86asm-1.0.2.jar
json-20090211.jar
jsr305-3.0.2.jar
metrics-core-4.1.18.jar
native-protocol-1.5.0.jar
netty-buffer-4.1.60.Final.jar
netty-codec-4.1.60.Final.jar
netty-common-4.1.60.Final.jar
netty-handler-4.1.60.Final.jar
netty-resolver-4.1.60.Final.jar
netty-transport-4.1.60.Final.jar
reactive-streams-1.0.3.jar
slf4j-api-1.7.26.jar
spotbugs-annotations-3.1.12.jar

10.2.7.8.2 Cassandra Datastax Java Driver 4.9.0

asm-7.1.jar
asm-analysis-7.1.jar
asm-commons-7.1.jar
asm-tree-7.1.jar
asm-util-7.1.jar
commons-collections-3.2.2.jar
commons-configuration-1.10.jar
commons-lang-2.6.jar
commons-lang3-3.8.1.jar
config-1.3.4.jar
esri-geometry-api-1.2.1.jar
gremlin-core-3.4.8.jar
gremlin-shaded-3.4.8.jar
HdrHistogram-2.1.11.jar
jackson-annotations-2.11.0.jar
jackson-core-2.11.0.jar
jackson-core-asl-1.9.12.jar
jackson-databind-2.11.0.jar
java-driver-core-4.9.0.jar
java-driver-query-builder-4.9.0.jar
java-driver-shaded-guava-25.1-jre-graal-sub-1.jar
javapoet-1.8.0.jar
javatuples-1.2.jar
jcip-annotations-1.0-1.jar
jcl-over-slf4j-1.7.25.jar
jffi-1.2.19.jar
jffi-1.2.19-native.jar
jnr-a64asm-1.0.0.jar
jnr-constants-0.9.12.jar
jnr-ffi-2.1.10.jar
jnr-posix-3.0.50.jar
jnr-x86asm-1.0.2.jar
json-20090211.jar
jsr305-3.0.2.jar
metrics-core-4.0.5.jar
native-protocol-1.4.11.jar
netty-buffer-4.1.51.Final.jar
netty-codec-4.1.51.Final.jar
netty-common-4.1.51.Final.jar
netty-handler-4.1.51.Final.jar
netty-resolver-4.1.51.Final.jar
netty-transport-4.1.51.Final.jar
reactive-streams-1.0.2.jar

slf4j-api-1.7.26.jar
spotbugs-annotations-3.1.12.jar
tinkergraph-gremlin-3.4.8.jar

10.2.8 Apache HBase

The HBase Handler is used to populate HBase tables from existing Oracle GoldenGate supported sources.

This chapter describes how to use the HBase Handler.

- [Overview](#)
- [Detailed Functionality](#)
- [Setting Up and Running the HBase Handler](#)
- [Security](#)
- [Metadata Change Events](#)

The HBase Handler seamlessly accommodates metadata change events including adding a column or dropping a column. The only requirement is that the source trail file contains the metadata.
- [Additional Considerations](#)
- [Troubleshooting the HBase Handler](#)

Troubleshooting of the HBase Handler begins with the contents for the Java log4j file. Follow the directions in the Java Logging Configuration to configure the runtime to correctly generate the Java log4j log file.
- [HBase Handler Client Dependencies](#)

What are the dependencies for the HBase Handler to connect to Apache HBase databases?

10.2.8.1 Overview

HBase is an open source Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) application that emulates much of the functionality of a relational database management system (RDBMS). Hadoop is specifically designed to store large amounts of unstructured data. Conversely, data stored in databases and replicated through Oracle GoldenGate is highly structured. HBase provides a way to maintain the important structure of data while taking advantage of the horizontal scaling that is offered by the Hadoop Distributed File System (HDFS).

10.2.8.2 Detailed Functionality

The HBase Handler takes operations from the source trail file and creates corresponding tables in HBase, and then loads change capture data into those tables.

HBase Table Names

Table names created in an HBase map to the corresponding table name of the operation from the source trail file. Table name is case-sensitive.

HBase Table Namespace

For two-part table names (schema name and table name), the schema name maps to the HBase table namespace. For a three-part table name like Catalog.Schema.MyTable, the create

HBase namespace would be `Catalog_Schema`. HBase table namespaces are case sensitive. A null schema name is supported and maps to the default HBase namespace.

HBase Row Key

HBase has a similar concept to the database primary keys, called the HBase row key. The HBase row key is the unique identifier for a table row. HBase only supports a single row key per row and it cannot be empty or null. The HBase Handler maps the primary key value into the HBase row key value. If the source table has multiple primary keys, then the primary key values are concatenated, separated by a pipe delimiter (`|`). You can configure the HBase row key delimiter.

If there's no primary/unique keys at the source table, then Oracle GoldenGate behaves as follows:

- If `KEYCOLS` is specified, then it constructs the key based on the specifications defined in the `KEYCOLS` clause.
- If `KEYCOLS` is not specified, then it constructs a key based on the concatenation of all eligible columns of the table.

The result is that the value of every column is concatenated to generate the HBase rowkey. However, this is not a good practice.

Workaround: Use the replicat mapping statement to identify one or more primary key columns. For example: `MAP QASOURCE.TCUSTORD, TARGET QASOURCE.TCUSTORD, KEYCOLS (CUST_CODE);`

HBase Column Family

HBase has the concept of a column family. A column family is a way to group column data. Only a single column family is supported. Every HBase column must belong to a single column family. The HBase Handler provides a single column family per table that defaults to `cf`. You can configure the column family name. However, after a table is created with a specific column family name, you cannot reconfigure the column family name in the HBase example, without first modifying or dropping the table results in an abend of the Oracle GoldenGateReplicat processes.

10.2.8.3 Setting Up and Running the HBase Handler

HBase must run either collocated with the HBase Handler process or on a machine that can connect from the network that is hosting the HBase Handler process. The underlying HDFS single instance or clustered instance serving as the repository for HBase data must also run.

Instructions for configuring the HBase Handler components and running the handler are described in this topic.

- [Classpath Configuration](#)
- [HBase Handler Configuration](#)
- [Sample Configuration](#)
- [Performance Considerations](#)

10.2.8.3.1 Classpath Configuration

For the HBase Handler to connect to HBase and stream data, the `hbase-site.xml` file and the HBase client jars must be configured in `gg.classpath` variable. The HBase client jars must match the version of HBase to which the HBase Handler is connecting. The HBase client jars

are not shipped with the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) product.

[HBase Handler Client Dependencies](#) lists the required HBase client jars by version.

The default location of the `hbase-site.xml` file is `HBase_Home/conf`.

The default location of the HBase client JARs is `HBase_Home/lib/*`.

If the HBase Handler is running on Windows, then follow the Windows classpathing syntax.

The `gg.classpath` must be configured exactly as described. The path to the `hbase-site.xml` file must contain only the path with no wild card appended. The inclusion of the `*` wildcard in the path to the `hbase-site.xml` file will cause it to be inaccessible. Conversely, the path to the dependency jars must include the `(*)` wildcard character in order to include all the jar files in that directory, in the associated classpath. Do not use `*.jar`. The following is an example of a correctly configured `gg.classpath` variable:

```
gg.classpath=/var/lib/hbase/lib/*:/var/lib/hbase/conf
```

10.2.8.3.2 HBase Handler Configuration

The following are the configurable values for the HBase Handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the HBase Handler, you must first configure the handler type by specifying `gg.handler.jdbc.type=hbase` and the other HBase properties as follows:

Table 10-9 HBase Handler Configuration Properties

Properties	Require d/ Optional	Legal Values	Default	Explanation
<code>gg.handlerlist</code>	Required	Any string.	None	Provides a name for the HBase Handler. The HBase Handler name is then becomes part of the property names listed in this table.
<code>gg.handler.name</code>	Required	<code>hbase</code> .	None	Selects the HBase Handler for streaming change data capture into HBase.
<code>gg.handler.name.hBaseColumnFamilyName</code>	Optional	Any string legal for an HBase column family name.	<code>cf</code>	Column family is a grouping mechanism for columns in HBase. The HBase Handler only supports a single column family.
<code>gg.handler.name.HBase20Compatible</code>	Optional	<code>true</code> <code>false</code>	<code>false</code> (HBase 1.0 compatible)	HBase 2.x removed methods and changed object hierarchies. The result is that it broke the binary compatibility with HBase 1.x. Set this property to <code>true</code> to correctly interface with HBase 2.x, otherwise HBase 1.x compatibility is used.
<code>gg.handler.name.includeTokens</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	Using <code>true</code> indicates that token values are included in the output to HBase. Using <code>false</code> means token values are not to be included.
<code>gg.handler.name.keyValueDelimiter</code>	Optional	Any string.	<code>=</code>	Provides a delimiter between key values in a map. For example, <code>key=value, key1=value1, key2=value2</code> . Tokens are mapped values. Configuration value supports CDATA[] wrapping.

Table 10-9 (Cont.) HBase Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.keyValuePairDelimiter</code>	Optional	Any string.	,	Provides a delimiter between key value pairs in a map. For example, <code>key=value, key1=value1, key2=value2key=value, key1=value1, key2=value2</code> . Tokens are mapped values. Configuration value supports CDATA[] wrapping.
<code>gg.handler.name.encoding</code>	Optional	Any encoding name or alias supported by Java. ¹ For a list of supported options, see https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html .	The native system encoding of the machine hosting the Oracle GoldenGate process	Determines the encoding of values written the HBase. HBase values are written as bytes.
<code>gg.handler.name.pkUpdateHandling</code>	Optional	<code>abend</code> <code>update</code> <code>delete-insert</code>	<code>abend</code>	Provides configuration for how the HBase Handler should handle update operations that change a primary key. Primary key operations can be problematic for the HBase Handler and require special consideration by you. <code>abend</code>: indicates the process will end abnormally. <code>update</code>: indicates the process will treat this as a normal update <code>delete-insert</code>: indicates the process will treat this as a delete and an insert. The full before image is required for this feature to work properly. This can be achieved by using full supplemental logging in Oracle Database. Without full before and after row images the insert data will be incomplete.
<code>gg.handler.name.nullValueRepresentation</code>	Optional	Any string.	NULL	Allows you to configure what will be sent to HBase in the case of a NULL column value. The default is NULL. Configuration value supports CDATA[] wrapping.
<code>gg.handler.name.authType</code>	Optional	<code>kerberos</code>	None	Setting this property to <code>kerberos</code> enables Kerberos authentication.

Table 10-9 (Cont.) HBase Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.kerberosKeytabFile</code>	Optional (Required if <code>authType=kerberos</code>)	Relative or absolute path to a Kerberos keytab file.	-	The keytab file allows the HDFS Handler to access a password to perform a <code>kinit</code> operation for Kerberos security.
<code>gg.handler.name.kerberosPrincipal</code>	Optional (Required if <code>authType=kerberos</code>)	A legal Kerberos principal name (for example, <code>user/FQDN@MY.REALM</code>)	-	The Kerberos principal name for Kerberos authentication.
<code>gg.handler.name.rowkeyDelimiter</code>	Optional	Any string/		Configures the delimiter between primary key values from the source table when generating the HBase rowkey. This property supports <code>CDATA[]</code> wrapping of the value to preserve whitespace if the user wishes to delimit incoming primary key values with a character or characters determined to be whitespace.
<code>gg.handler.name.setHBaseOperationTimestamp</code>	Optional	<code>true</code> <code>false</code>	<code>true</code>	Set to <code>true</code> to set the timestamp for HBase operations in the HBase Handler instead of allowing HBase to assign the timestamps on the server side. This property can be used to solve the problem of a row delete followed by an immediate reinsert of the row not showing up in HBase, see HBase Handler Delete-Insert Problem .
<code>gg.handler.name.omitNullValues</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	Set to <code>true</code> to omit null fields from being written.
<code>gg.handler.name.metaColumnsTemplate</code>	Optional	A legal string	None	A legal string specifying the metaColumns to be included. For more information, see Metacolumn Keywords .

¹ See *Java Internalization Support* at <https://docs.oracle.com/javase/8/docs/technotes/guides/intl/>.

10.2.8.3.3 Sample Configuration

The following is a sample configuration for the HBase Handler from the Java Adapter properties file:

```
gg.handlerlist=hbase
gg.handler.hbase.type=hbase
gg.handler.hbase.mode=tx
gg.handler.hbase.hBaseColumnFamilyName=cf
gg.handler.hbase.includeTokens=true
gg.handler.hbase.keyValueDelimiter=CDATA[=]
gg.handler.hbase.keyValuePairDelimiter=CDATA[, ]
gg.handler.hbase.encoding=UTF-8
```

```
gg.handler.hbase.pkUpdateHandling=abend  
gg.handler.hbase.nullValueRepresentation=CDATA[NULL]  
gg.handler.hbase.authType=none
```

10.2.8.3.4 Performance Considerations

At each transaction commit, the HBase Handler performs a flush call to flush any buffered data to the HBase region server. This must be done to maintain write durability. Flushing to the HBase region server is an expensive call and performance can be greatly improved by using the Replicat GROUPTRANSOPS parameter to group multiple smaller transactions in the source trail file into a larger single transaction applied to HBase. You can use Replicat base-batching by adding the configuration syntax in the Replicat configuration file.

Operations from multiple transactions are grouped together into a larger transaction, and it is only at the end of the grouped transaction that transaction is committed.

10.2.8.4 Security

You can secure HBase connectivity using Kerberos authentication. Follow the associated documentation for the HBase release to secure the HBase cluster. The HBase Handler can connect to Kerberos secured clusters. The HBase hbase-site.xml must be in handlers classpath with the hbase.security.authentication property set to kerberos and hbase.security.authorization property set to true.

You have to include the directory containing the HDFS core-site.xml file in the classpath. Kerberos authentication is performed using the Hadoop UserGroupInformation class. This class relies on the Hadoop configuration property hadoop.security.authentication being set to kerberos to successfully perform the kinit command.

Additionally, you must set the following properties in the HBase Handler Java configuration file:

```
gg.handler.{name}.authType=kerberos  
gg.handler.{name}.kerberosPrincipalName={legal Kerberos principal name}  
gg.handler.{name}.kerberosKeytabFile={path to a keytab file that contains the password  
for the Kerberos principal so that the Oracle GoldenGate HDFS handler can  
programmatically perform the Kerberos kinit operations to obtain a Kerberos ticket}.
```

You may encounter the inability to decrypt the Kerberos password from the keytab file. This causes the Kerberos authentication to fall back to interactive mode which cannot work because it is being invoked programmatically. The cause of this problem is that the Java Cryptography Extension (JCE) is not installed in the Java Runtime Environment (JRE). Ensure that the JCE is loaded in the JRE, see <http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html>.

10.2.8.5 Metadata Change Events

The HBase Handler seamlessly accommodates metadata change events including adding a column or dropping a column. The only requirement is that the source trail file contains the metadata.

10.2.8.6 Additional Considerations

Classpath issues are common during the initial setup of the HBase Handler. The typical indicators are occurrences of the ClassNotFoundException in the Java log4j log file. The HBase client jars do not ship with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA). You must resolve the required HBase client jars. [HBase Handler Client Dependencies](#) includes a list of HBase client jars for each supported version. Either the hbase-

site.xml or one or more of the required client JARS are not included in the classpath. For instructions on configuring the classpath of the HBase Handler, see [Classpath Configuration](#).

10.2.8.7 Troubleshooting the HBase Handler

Troubleshooting of the HBase Handler begins with the contents for the Java log4j file. Follow the directions in the Java Logging Configuration to configure the runtime to correctly generate the Java log4j log file.

- [Java Classpath](#)
- [HBase Connection Properties](#)
- [Logging of Handler Configuration](#)
- [HBase Handler Delete-Insert Problem](#)

10.2.8.7.1 Java Classpath

Issues with the Java classpath are common. A `ClassNotFoundException` in the Java log4j log file indicates a classpath problem. You can use the Java log4j log file to troubleshoot this issue. Setting the log level to `DEBUG` logs each of the jars referenced in the `gg.classpath` object to the log file. You can make sure that all of the required dependency jars are resolved by enabling `DEBUG` level logging, and then searching the log file for messages like the following:

```
2015-09-29 13:04:26 DEBUG ConfigClassPath:74 - ...adding to classpath:
url="file:/ggwork/hbase/hbase-1.0.1.1/lib/hbase-server-1.0.1.1.jar"
```

10.2.8.7.2 HBase Connection Properties

The contents of the HDFS `hbase-site.xml` file (including default settings) are output to the Java log4j log file when the logging level is set to `DEBUG` or `TRACE`. This file shows the connection properties to HBase. Search for the following in the Java log4j log file.

```
2015-09-29 13:04:27 DEBUG HBaseWriter:449 - Begin - HBase configuration object contents
for connection troubleshooting.
Key: [hbase.auth.token.max.lifetime] Value: [604800000].
```

Commonly, for the `hbase-site.xml` file is not included in the classpath or the path to the `hbase-site.xml` file is incorrect. In this case, the HBase Handler cannot establish a connection to HBase, and the Oracle GoldenGate process abends. The following error is reported in the Java log4j log.

```
2015-09-29 12:49:29 ERROR HBaseHandler:207 - Failed to initialize the HBase handler.
org.apache.hadoop.hbase.ZooKeeperConnectionException: Can't connect to ZooKeeper
```

Verify that the classpath correctly includes the `hbase-site.xml` file and that HBase is running.

10.2.8.7.3 Logging of Handler Configuration

The Java log4j log file contains information on the configuration state of the HBase Handler. This information is output at the `INFO` log level. The following is a sample output:

```
2015-09-29 12:45:53 INFO HBaseHandler:194 - **** Begin HBase Handler - Configuration
Summary ****
Mode of operation is set to tx.
HBase data will be encoded using the native system encoding.
In the event of a primary key update, the HBase Handler will ABEND.
HBase column data will use the column family name [cf].
The HBase Handler will not include tokens in the HBase data.
```


The HBase Handler has been configured to use [=] as the delimiter between keys and values.

The HBase Handler has been configured to use [,] as the delimiter between key values pairs.

The HBase Handler has been configured to output [NULL] for null values.

Hbase Handler Authentication type has been configured to use [none]

10.2.8.7.4 HBase Handler Delete-Insert Problem

If you are using the HBase Handler with the `gg.handler.name.setHbaseOperationTimestamp=false` configuration property, then the source database may get out of sync with data in the HBase tables. This is caused by the deletion of a row followed by the immediate reinsertion of the row. HBase creates a tombstone marker for the delete that is identified by a specific timestamp. This tombstone marker marks any row records in HBase with the same row key as deleted that have a timestamp before or the same as the tombstone marker. This can occur when the deleted row is immediately reinserted. The insert operation can inadvertently have the same timestamp as the delete operation so the delete operation causes the subsequent insert operation to incorrectly appear as deleted.

To work around this issue, you need to set the

`gg.handler.name.setHbaseOperationTimestamp=true`, which does two things:

- Sets the timestamp for row operations in the HBase Handler.
- Detection of a delete-insert operation that ensures that the insert operation has a timestamp that is after the insert.

The default for `gg.handler.name.setHbaseOperationTimestamp` is `true`, which means that the HBase server supplies the timestamp for a row. This prevents the HBase delete-reinsert out-of-sync problem.

Setting the row operation timestamp in the HBase Handler can have these consequences:

1. Since the timestamp is set on the client side, this could create problems if multiple applications are feeding data to the same HBase table.
2. If delete and reinsert is a common pattern in your use case, then the HBase Handler has to increment the timestamp 1 millisecond each time this scenario is encountered.

Processing cannot be allowed to get too far into the future so the HBase Handler only allows the timestamp to increment 100 milliseconds into the future before it attempts to wait the process so that the client side HBase operation timestamp and real time are back in sync. When a delete-insert is used instead of an update in the source database so this sync scenario would be quite common. Processing speeds may be affected by not allowing the HBase timestamp to go over 100 milliseconds into the future if this scenario is common.

10.2.8.8 HBase Handler Client Dependencies

What are the dependencies for the HBase Handler to connect to Apache HBase databases?

The maven central repository artifacts for HBase databases are:

- **Maven groupId:** `org.apache.hbase`
- **Maven artifactId:** `hbase-client`
- **Maven version:** the HBase version numbers listed for each section

The `hbase-client-x.x.x.jar` file is not distributed with Apache HBase, nor is it mandatory to be in the classpath. The `hbase-client-x.x.x.jar` file is an empty Maven project whose purpose of aggregating all of the HBase client dependencies.

- [HBase 2.4.4](#)
- [HBase 2.3.3](#)
- [HBase 2.2.0](#)
- [HBase 2.1.5](#)
- [HBase 2.0.5](#)
- [HBase 1.4.10](#)
- [HBase 1.3.3](#)
- [HBase 1.2.5](#)
- [HBase 1.1.1](#)
- [HBase 1.0.1.1](#)

10.2.8.8.1 HBase 2.4.4

apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
audience-annotations-0.5.0.jar
avro-1.7.7.jar
commons-beanutils-1.9.4.jar
commons-cli-1.2.jar
commons-codec-1.13.jar
commons-collections-3.2.2.jar
commons-compress-1.19.jar
commons-configuration-1.6.jar
commons-crypto-1.0.0.jar
commons-digester-1.8.jar
commons-io-2.6.jar
commons-lang-2.6.jar
commons-lang3-3.9.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.7.1.jar
curator-framework-2.7.1.jar
curator-recipes-2.7.1.jar
error_prone_annotations-2.3.4.jar
gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-2.10.0.jar
hadoop-auth-2.10.0.jar
hadoop-common-2.10.0.jar
hbase-client-2.4.4.jar
hbase-common-2.4.4.jar
hbase-hadoop2-compat-2.4.4.jar
hbase-hadoop-compat-2.4.4.jar
hbase-logging-2.4.4.jar
hbase-metrics-2.4.4.jar
hbase-metrics-api-2.4.4.jar
hbase-protocol-2.4.4.jar
hbase-protocol-shaded-2.4.4.jar
hbase-shaded-gson-3.4.1.jar
hbase-shaded-miscellaneous-3.4.1.jar
hbase-shaded-netty-3.4.1.jar
hbase-shaded-protobuf-3.4.1.jar
htrace-core4-4.2.0-incubating.jar

httpclient-4.5.2.jar
httpcore-4.4.4.jar
jackson-core-asl-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
javax.activation-api-1.2.0.jar
jcip-annotations-1.0-1.jar
jcodings-1.0.55.jar
jdk.tools-1.8.jar
jetty-sslengine-6.1.26.jar
joni-2.1.31.jar
jsch-0.1.54.jar
jsr305-3.0.0.jar
log4j-1.2.17.jar
metrics-core-3.2.6.jar
netty-buffer-4.1.45.Final.jar
netty-codec-4.1.45.Final.jar
netty-common-4.1.45.Final.jar
netty-handler-4.1.45.Final.jar
netty-resolver-4.1.45.Final.jar
netty-transport-4.1.45.Final.jar
netty-transport-native-epoll-4.1.45.Final.jar
netty-transport-native-unix-common-4.1.45.Final.jar
nimbus-jose-jwt-4.41.1.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.7.30.jar
slf4j-log4j12-1.7.25.jar
snappy-java-1.0.5.jar
stax2-api-3.1.4.jar
woodstox-core-5.0.3.jar
xmlenc-0.52.jar
zookeeper-3.5.7.jar
zookeeper-jute-3.5.7.jar

10.2.8.8.2 HBase 2.3.3

apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
audience-annotations-0.5.0.jar
avro-1.7.7.jar
commons-beanutils-1.9.4.jar
commons-cli-1.2.jar
commons-codec-1.13.jar
commons-collections-3.2.2.jar
commons-compress-1.19.jar
commons-configuration-1.6.jar
commons-crypto-1.0.0.jar
commons-digester-1.8.jar
commons-io-2.6.jar
commons-lang-2.6.jar
commons-lang3-3.9.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.7.1.jar
curator-framework-2.7.1.jar
curator-recipes-2.7.1.jar
error_prone_annotations-2.3.4.jar
gson-2.2.4.jar
guava-11.0.2.jar

hadoop-annotations-2.10.0.jar
hadoop-auth-2.10.0.jar
hadoop-common-2.10.0.jar
hbase-client-2.3.3.jar
hbase-common-2.3.3.jar
hbase-hadoop2-compat-2.3.3.jar
hbase-hadoop-compat-2.3.3.jar
hbase-logging-2.3.3.jar
hbase-metrics-2.3.3.jar
hbase-metrics-api-2.3.3.jar
hbase-protocol-2.3.3.jar
hbase-protocol-shaded-2.3.3.jar
hbase-shaded-gson-3.3.0.jar
hbase-shaded-miscellaneous-3.3.0.jar
hbase-shaded-netty-3.3.0.jar
hbase-shaded-protobuf-3.3.0.jar
htrace-core4-4.2.0-incubating.jar
httpclient-4.5.2.jar
httpcore-4.4.4.jar
jackson-core-asl-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
javax.activation-api-1.2.0.jar
jcip-annotations-1.0-1.jar
jcodings-1.0.18.jar
jdk.tools-1.8.jar

10.2.8.3 HBase 2.2.0

apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
audience-annotations-0.5.0.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.10.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-crypto-1.0.0.jar
commons-digester-1.8.jar
commons-io-2.5.jar
commons-lang-2.6.jar
commons-lang3-3.6.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.7.1.jar
curator-framework-2.7.1.jar
curator-recipes-2.7.1.jar
error_prone_annotations-2.3.3.jar
findbugs-annotations-1.3.9-1.jar
gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-2.8.5.jar
hadoop-auth-2.8.5.jar
hadoop-common-2.8.5.jar
hamcrest-core-1.3.jar
hbase-client-2.2.0.jar
hbase-common-2.2.0.jar

hbase-hadoop2-compat-2.2.0.jar
hbase-hadoop-compat-2.2.0.jar
hbase-metrics-2.2.0.jar
hbase-metrics-api-2.2.0.jar
hbase-protocol-2.2.0.jar
hbase-protocol-shaded-2.2.0.jar
hbase-shaded-miscellaneous-2.2.1.jar
hbase-shaded-netty-2.2.1.jar
hbase-shaded-protobuf-2.2.1.jar
htrace-core4-4.2.0-incubating.jar
httpclient-4.5.2.jar
httpcore-4.4.4.jar
jackson-core-asl-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jcip-annotations-1.0-1.jar
jcodings-1.0.18.jar
jdk.tools-1.8.jar
jetty-sslengine-6.1.26.jar
joni-2.1.11.jar
jsch-0.1.54.jar
jsr305-3.0.0.jar
junit-4.12.jar
log4j-1.2.17.jar
metrics-core-3.2.6.jar
nimbus-jose-jwt-4.4.1.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.7.25.jar
slf4j-log4j12-1.6.1.jar
snappy-java-1.0.4.1.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.10.jar

10.2.8.8.4 HBase 2.1.5

apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
audience-annotations-0.5.0.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.10.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-crypto-1.0.0.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.5.jar
commons-lang-2.6.jar
commons-lang3-3.6.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.7.1.jar
curator-framework-2.7.1.jar
curator-recipes-2.7.1.jar
findbugs-annotations-1.3.9-1.jar

gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-2.7.7.jar
hadoop-auth-2.7.7.jar
hadoop-common-2.7.7.jar
hamcrest-core-1.3.jar
hbase-client-2.1.5.jar
hbase-common-2.1.5.jar
hbase-hadoop2-compat-2.1.5.jar
hbase-hadoop-compat-2.1.5.jar
hbase-metrics-2.1.5.jar
hbase-metrics-api-2.1.5.jar
hbase-protocol-2.1.5.jar
hbase-protocol-shaded-2.1.5.jar
hbase-shaded-miscellaneous-2.1.0.jar
hbase-shaded-netty-2.1.0.jar
hbase-shaded-protobuf-2.1.0.jar
htrace-core-3.1.0-incubating.jar
htrace-core4-4.2.0-incubating.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-annotations-2.9.0.jar
jackson-core-2.9.2.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.9.2.jar
jackson-mapper-asl-1.9.13.jar
jcodings-1.0.18.jar
jdk.tools-1.8.jar
jetty-sslengine-6.1.26.jar
joni-2.1.11.jar
jsch-0.1.54.jar
jsr305-3.0.0.jar
junit-4.12.jar
log4j-1.2.17.jar
metrics-core-3.2.6.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.7.25.jar
slf4j-log4j12-1.6.1.jar
snappy-java-1.0.4.1.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.10.jar

10.2.8.8.5 HBase 2.0.5

apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
audience-annotations-0.5.0.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.10.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-crypto-1.0.0.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar

commons-io-2.5.jar
commons-lang-2.6.jar
commons-lang3-3.6.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.7.1.jar
curator-framework-2.7.1.jar
curator-recipes-2.7.1.jar
findbugs-annotations-1.3.9-1.jar
gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-2.7.7.jar
hadoop-auth-2.7.7.jar
hadoop-common-2.7.7.jar
hamcrest-core-1.3.jar
hbase-client-2.0.5.jar
hbase-common-2.0.5.jar
hbase-hadoop2-compat-2.0.5.jar
hbase-hadoop-compat-2.0.5.jar
hbase-metrics-2.0.5.jar
hbase-metrics-api-2.0.5.jar
hbase-protocol-2.0.5.jar
hbase-protocol-shaded-2.0.5.jar
hbase-shaded-miscellaneous-2.1.0.jar
hbase-shaded-netty-2.1.0.jar
hbase-shaded-protobuf-2.1.0.jar
htrace-core4-4.2.0-incubating.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-annotations-2.9.0.jar
jackson-core-2.9.2.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.9.2.jar
jackson-mapper-asl-1.9.13.jar
jcodings-1.0.18.jar
jdk.tools-1.8.jar
jetty-sslengine-6.1.26.jar
joni-2.1.11.jar
jsch-0.1.54.jar
jsr305-3.0.0.jar
junit-4.12.jar
log4j-1.2.17.jar
metrics-core-3.2.1.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.7.25.jar
slf4j-log4j12-1.6.1.jar
snappy-java-1.0.4.1.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.10.jar

10.2.8.8.6 HBase 1.4.10

activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.7.jar
commons-beanutils-1.7.0.jar

commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.9.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.2.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.7.1.jar
curator-framework-2.7.1.jar
curator-recipes-2.7.1.jar
findbugs-annotations-1.3.9-1.jar
gson-2.2.4.jar
guava-12.0.1.jar
hadoop-annotations-2.7.4.jar
hadoop-auth-2.7.4.jar
hadoop-common-2.7.4.jar
hadoop-mapreduce-client-core-2.7.4.jar
hadoop-yarn-api-2.7.4.jar
hadoop-yarn-common-2.7.4.jar
hamcrest-core-1.3.jar
hbase-annotations-1.4.10.jar
hbase-client-1.4.10.jar
hbase-common-1.4.10.jar
hbase-protocol-1.4.10.jar
htrace-core-3.1.0-incubating.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jaxb-api-2.2.2.jar
jcodings-1.0.8.jar
jdk.tools-1.8.jar
jetty-sslengine-6.1.26.jar
jetty-util-6.1.26.jar
joni-2.1.2.jar
jsch-0.1.54.jar
jsr305-3.0.0.jar
junit-4.12.jar
log4j-1.2.17.jar
metrics-core-2.2.0.jar
netty-3.6.2.Final.jar
netty-all-4.1.8.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.6.1.jar
slf4j-log4j12-1.6.1.jar
snappy-java-1.0.5.jar
stax-api-1.0-2.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.10.jar

10.2.8.8.7 HBase 1.3.3

activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar

apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.9.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-el-1.0.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.2.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
findbugs-annotations-1.3.9-1.jar
guava-12.0.1.jar
hadoop-annotations-2.5.1.jar
hadoop-auth-2.5.1.jar
hadoop-common-2.5.1.jar
hadoop-mapreduce-client-core-2.5.1.jar
hadoop-yarn-api-2.5.1.jar
hadoop-yarn-common-2.5.1.jar
hamcrest-core-1.3.jar
hbase-annotations-1.3.3.jar
hbase-client-1.3.3.jar
hbase-common-1.3.3.jar
hbase-protocol-1.3.3.jar
htrace-core-3.1.0-incubating.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jaxb-api-2.2.2.jar
jcodings-1.0.8.jar
jdk.tools-1.6.jar
jetty-util-6.1.26.jar
joni-2.1.2.jar
jsch-0.1.42.jar
jsr305-1.3.9.jar
junit-4.12.jar
log4j-1.2.17.jar
metrics-core-2.2.0.jar
netty-3.6.2.Final.jar
netty-all-4.0.50.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.6.1.jar
slf4j-log4j12-1.6.1.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar

10.2.8.8.8 HBase 1.2.5

activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.9.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-el-1.0.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.2.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
findbugs-annotations-1.3.9-1.jar
guava-12.0.1.jar
hadoop-annotations-2.5.1.jar
hadoop-auth-2.5.1.jar
hadoop-common-2.5.1.jar
hadoop-mapreduce-client-core-2.5.1.jar
hadoop-yarn-api-2.5.1.jar
hadoop-yarn-common-2.5.1.jar
hamcrest-core-1.3.jar
hbase-annotations-1.2.5.jar
hbase-client-1.2.5.jar
hbase-common-1.2.5.jar
hbase-protocol-1.2.5.jar
htrace-core-3.1.0-incubating.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jaxb-api-2.2.2.jar
jcodings-1.0.8.jar
jdk.tools-1.6.jar
jetty-util-6.1.26.jar
joni-2.1.2.jar
jsch-0.1.42.jar
jsr305-1.3.9.jar
junit-4.12.jar
log4j-1.2.17.jar
metrics-core-2.2.0.jar
netty-3.6.2.Final.jar
netty-all-4.0.23.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.6.1.jar
slf4j-log4j12-1.6.1.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xmlenc-0.52.jar

xz-1.0.jar
zookeeper-3.4.6.jar

10.2.8.8.9 HBase 1.1.1

HBase 1.1.1 is effectively the same as HBase 1.1.0.1. You can substitute 1.1.0.1 in the libraries that are versioned as 1.1.1.

activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.9.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-el-1.0.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.2.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
findbugs-annotations-1.3.9-1.jar
guava-12.0.1.jar
hadoop-annotations-2.5.1.jar
hadoop-auth-2.5.1.jar
hadoop-common-2.5.1.jar
hadoop-mapreduce-client-core-2.5.1.jar
hadoop-yarn-api-2.5.1.jar
hadoop-yarn-common-2.5.1.jar
hamcrest-core-1.3.jar
hbase-annotations-1.1.1.jar
hbase-client-1.1.1.jar
hbase-common-1.1.1.jar
hbase-protocol-1.1.1.jar
htrace-core-3.1.0-incubating.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jaxb-api-2.2.2.jar
jcodings-1.0.8.jar
jdk.tools-1.7.jar
jetty-util-6.1.26.jar
joni-2.1.2.jar
jsch-0.1.42.jar
jsr305-1.3.9.jar
junit-4.11.jar
log4j-1.2.17.jar
netty-3.6.2.Final.jar
netty-all-4.0.23.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.6.1.jar
slf4j-log4j12-1.6.1.jar

snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar

10.2.8.8.10 HBase 1.0.1.1

activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.9.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-el-1.0.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.2.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
findbugs-annotations-1.3.9-1.jar
guava-12.0.1.jar
hadoop-annotations-2.5.1.jar
hadoop-auth-2.5.1.jar
hadoop-common-2.5.1.jar
hadoop-mapreduce-client-core-2.5.1.jar
hadoop-yarn-api-2.5.1.jar
hadoop-yarn-common-2.5.1.jar
hamcrest-core-1.3.jar
hbase-annotations-1.0.1.1.jar
hbase-client-1.0.1.1.jar
hbase-common-1.0.1.1.jar
hbase-protocol-1.0.1.1.jar
htrace-core-3.1.0-incubating.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.8.8.jar
jackson-mapper-asl-1.8.8.jar
jaxb-api-2.2.2.jar
jcodings-1.0.8.jar
jdk.tools-1.7.jar
jetty-util-6.1.26.jar
joni-2.1.2.jar
jsch-0.1.42.jar
jsr305-1.3.9.jar
junit-4.11.jar
log4j-1.2.17.jar
netty-3.6.2.Final.jar
netty-all-4.0.23.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.6.1.jar
slf4j-log4j12-1.6.1.jar

```
snappy-java-1.0.4.1.jar  
stax-api-1.0-2.jar  
xmlenc-0.52.jar  
xz-1.0.jar  
zookeeper-3.4.6.jar
```

10.2.9 Apache HDFS

The HDFS Handler is designed to stream change capture data into the Hadoop Distributed File System (HDFS).

This chapter describes how to use the HDFS Handler.

- [Overview](#)
- [Writing into HDFS in SequenceFile Format](#)
The HDFS SequenceFile is a flat file consisting of binary key and value pairs. You can enable writing data in SequenceFile format by setting the `gg.handler.name.format` property to `sequencefile`.
- [Setting Up and Running the HDFS Handler](#)
- [Writing in HDFS in Avro Object Container File Format](#)
- [Generating HDFS File Names Using Template Strings](#)
- [Metadata Change Events](#)
- [Partitioning](#)
The partitioning functionality uses the template mapper functionality to resolve partitioning strings. The result is that the you have more control in how to partition source trail data. Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 21c release, all the keywords that are supported by the templating functionality are supported in HDFS partitioning.
- [HDFS Additional Considerations](#)
- [Best Practices](#)
- [Troubleshooting the HDFS Handler](#)
Troubleshooting of the HDFS Handler begins with the contents for the Java `log4j` file. Follow the directions in the Java Logging Configuration to configure the runtime to correctly generate the Java `log4j` log file.
- [HDFS Handler Client Dependencies](#)

10.2.9.1 Overview

The HDFS is the primary file system for Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA). Hadoop is typically installed on multiple machines that work together as a Hadoop cluster. Hadoop allows you to store very large amounts of data in the cluster that is horizontally scaled across the machines in the cluster. You can then perform analytics on that data using a variety of GG for DAA applications.

10.2.9.2 Writing into HDFS in SequenceFile Format

The HDFS SequenceFile is a flat file consisting of binary key and value pairs. You can enable writing data in SequenceFile format by setting the `gg.handler.name.format` property to `sequencefile`.

The key part of the record is set to null, and the actual data is set in the value part. For information about Hadoop SequenceFile, see <https://cwiki.apache.org/confluence/display/HADOOP2/SequenceFile>.

- [Integrating with Hive](#)
- [Understanding the Data Format](#)

10.2.9.2.1 Integrating with Hive

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) release does not include a Hive storage handler because the HDFS Handler provides all of the necessary Hive functionality.

You can create a Hive integration to create tables and update table definitions in case of DDL events. This is limited to data formatted in Avro Object Container File format. For more information, see [Writing in HDFS in Avro Object Container File Format](#) and [HDFS Handler Configuration](#).

For Hive to consume sequence files, the DDL creates Hive tables including STORED as sequencefile. The following is a sample create table script:

```
CREATE EXTERNAL TABLE table_name (  
  col1 string,  
  ...  
  ...  
  col2 string)  
ROW FORMAT DELIMITED  
STORED as sequencefile  
LOCATION '/path/to/hdfs/file';
```

Note

If files are intended to be consumed by Hive, then the `gg.handler.name.partitionByTable` property should be set to true.

10.2.9.2.2 Understanding the Data Format

The data written in the value part of each record and is in delimited text format. All of the options described in the [Using the Delimited Text Row Formatter](#) section are applicable to HDFS SequenceFile when writing data to it.

For example:

```
gg.handler.name.format=sequencefile  
gg.handler.name.format.includeColumnNames=true  
gg.handler.name.format.includeOpType=true  
gg.handler.name.format.includeCurrentTimestamp=true  
gg.handler.name.format.updateOpKey=U
```

10.2.9.3 Setting Up and Running the HDFS Handler

To run the HDFS Handler, a Hadoop single instance or Hadoop cluster must be installed, running, and network-accessible from the machine running the HDFS Handler. Apache Hadoop is open source and you can download it from:

<http://hadoop.apache.org/>

Follow the Getting Started links for information on how to install a single-node cluster (for pseudo-distributed operation mode) or a clustered setup (for fully-distributed operation mode).

Instructions for configuring the HDFS Handler components and running the handler are described in the following sections.

- [Classpath Configuration](#)
- [HDFS Handler Configuration](#)
- [Review a Sample Configuration](#)
- [Performance Considerations](#)
- [Security](#)

10.2.9.3.1 Classpath Configuration

For the HDFS Handler to connect to HDFS and run, the HDFS `core-site.xml` file and the HDFS client jars must be configured in `gg.classpath` variable. The HDFS client jars must match the version of HDFS that the HDFS Handler is connecting. For a list of the required client jar files by release, see [HDFS Handler Client Dependencies](#).

The default location of the `core-site.xml` file is `Hadoop_Home/etc/hadoop`

The default locations of the HDFS client jars are the following directories:

`Hadoop_Home/share/hadoop/common/lib/*`

`Hadoop_Home/share/hadoop/common/*`

`Hadoop_Home/share/hadoop/hdfs/lib/*`

`Hadoop_Home/share/hadoop/hdfs/*`

The `gg.classpath` must be configured exactly as shown. The path to the `core-site.xml` file must contain the path to the directory containing the `core-site.xml` file with no wildcard appended. If you include a (*) wildcard in the path to the `core-site.xml` file, the file is not picked up. Conversely, the path to the dependency jars must include the (*) wildcard character in order to include all the jar files in that directory in the associated classpath. Do not use `*.jar`.

The following is an example of a correctly configured `gg.classpath` variable:

```
gg.classpath=/ggwork/hadoop/hadoop-2.6.0/etc/hadoop:/ggwork/hadoop/hadoop-2.6.0/share/hadoop/common/lib/*:/ggwork/hadoop/hadoop-2.6.0/share/hadoop/common/*:/ggwork/hadoop/hadoop-2.6.0/share/hadoop/hdfs/*:/ggwork/hadoop/hadoop-2.6.0/share/hadoop/hdfs/lib/*
```

The HDFS configuration file `hdfs-site.xml` must also be in the classpath if Kerberos security is enabled. By default, the `hdfs-site.xml` file is located in the `Hadoop_Home/etc/hadoop` directory. If the HDFS Handler is not colocated with Hadoop, either or both files can be copied to another machine.

10.2.9.3.2 HDFS Handler Configuration

The following are the configurable values for the HDFS Handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the HDFS Handler, you must first configure the handler type by specifying `gg.handler.name.type=hdfs` and the other HDFS properties as follows:

Property	Optional / Required	Legal Values	Default	Explanation
<code>gg.handlerlist</code>	Required	Any string	None	Provides a name for the HDFS Handler. The HDFS Handler name then becomes part of the property names listed in this table.
<code>gg.handler.name.type</code>	Required	<code>hdfs</code>	None	Selects the HDFS Handler for streaming change data capture into HDFS.
<code>gg.handler.name.mode</code>	Optional	<code>tx op</code>	<code>op</code>	Selects operation (<code>op</code>) mode or transaction (<code>tx</code>) mode for the handler. In almost all scenarios, transaction mode results in better performance.
<code>gg.handler.name.maxFileSize</code>	Optional	The default unit of measure is bytes. You can use <code>k</code> , <code>m</code> , or <code>g</code> to specify kilobytes, megabytes, or gigabytes. Examples of legal values include <code>10000</code> , <code>10k</code> , <code>100m</code> , <code>1.1g</code> .	<code>1g</code>	Selects the maximum file size of the created HDFS files.
<code>gg.handler.name.pathMappingTemplate</code>	Optional	Any legal templated string to resolve the target write directory in HDFS. Templates can contain a mix of constants and keywords which are dynamically resolved at runtime to generate the HDFS write directory.	<code>/ogg/\${toLowerCase[fullyQualifiedTableName]}</code>	You can use keywords interlaced with constants to dynamically generate the HDFS write directory at runtime, see Generating HDFS File Names Using Template Strings .
<code>gg.handler.name.fileRollInterval</code>	Optional	The default unit of measure is milliseconds. You can stipulate <code>ms</code> , <code>s</code> , <code>m</code> , <code>h</code> to signify milliseconds, seconds, minutes, or hours respectively. Examples of legal values include <code>10000</code> , <code>10000ms</code> , <code>10s</code> , <code>10m</code> , or <code>1.5h</code> . Values of <code>0</code> or less indicate that file rolling on time is turned off.	File rolling on time is off.	The timer starts when an HDFS file is created. If the file is still open when the interval elapses, then the file is closed. A new file is not immediately opened. New HDFS files are created on a just-in-time basis.

Property	Optional / Required	Legal Values	Default	Explanation
<code>gg.handler.name.inactivityRollInterval</code>	Optional	The default unit of measure is milliseconds. You can use ms, s, m, h to specify milliseconds, seconds, minutes, or hours. Examples of legal values include 10000, 10000ms, 10s, 10, 5m, or 1h. Values of 0 or less indicate that file inactivity rolling on time is turned off.	File inactivity rolling on time is off.	The timer starts from the latest write to an HDFS file. New writes to an HDFS file restart the counter. If the file is still open when the counter elapses, the HDFS file is closed. A new file is not immediately opened. New HDFS files are created on a just-in-time basis.
<code>gg.handler.name.fileNameMappingTemplate</code>	Optional	A string with resolvable keywords and constants used to dynamically generate HDFS file names at runtime.	<code>\$ {fullyQualifiedTableName}_\${groupName}_\${currentTimestamp}.txt</code>	You can use keywords interlaced with constants to dynamically generate unique HDFS file names at runtime, see Generating HDFS File Names Using Template Strings . File names typically follow the format, <code>\$ {fullyQualifiedTableName}_\${groupName}_\${currentTimestamp}.txt</code> .
<code>gg.handler.name.partitionByTable</code>	Optional	true false	true (data is partitioned by table)	Determines whether data written into HDFS must be partitioned by table. If set to true, then data for different tables are written to different HDFS files. If set to false, then data from different tables is interlaced in the same HDFS file. Must be set to true to use the Avro Object Container File Formatter. If set to false, a configuration exception occurs at initialization.
<code>gg.handler.name.rollOnMetadataChange</code>	Optional	true false	true (HDFS files are rolled on a metadata change event)	Determines whether HDFS files are rolled in the case of a metadata change. True means the HDFS file is rolled, false means the HDFS file is not rolled. Must be set to true to use the Avro Object Container File Formatter. If set to false, a configuration exception occurs at initialization.

Property	Optional / Required	Legal Values	Default	Explanation
<code>gg.handler.name.format</code>	Optional	<code>delimitedtext</code> <code>json</code> <code>json_row</code> <code>xml</code> <code>avro_row</code> <code>avro_op</code> <code>avro_row_ocf</code> <code>avro_op_ocf</code> <code>sequencefile</code>	<code>delimitedtext</code>	Selects the formatter for the HDFS Handler for how output data is formatted. <code>delimitedtext</code>: Delimited text <code>json</code>: JSON <code>json_row</code>: JSON output modeling row data <code>xml</code>: XML <code>avro_row</code>: Avro in row compact format <code>avro_op</code>: Avro in operation more verbose format. <code>avro_row_ocf</code>: Avro in the row compact format written into HDFS in the Avro Object Container File (OCF) format. <code>avro_op_ocf</code>: Avro in the more verbose format written into HDFS in the Avro Object Container File format. <code>sequencefile</code>: Delimited text written in sequence into HDFS is sequence file format.
<code>gg.handler.name.includeTokens</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	Set to <code>true</code> to include the tokens field and tokens key/values in the output. Set to <code>false</code> to suppress tokens output.
<code>gg.handler.name.partitioner.fully_qualified_table_name</code>	Optional	A mixture of templating keywords and constants to resolve a sub directory at runtime to partition the data.	-	The configuration resolves a sub directory or sub directories, which are appended to the resolved HDFS target path. These sub directories are used to partition the data. <code>gg.handler.name.partitionByTable</code> must be set to <code>true</code> .
<code>gg.handler.name.authType</code>	Optional	<code>kerberos</code>	<code>none</code>	Setting this property to <code>kerberos</code> enables Kerberos authentication.
<code>gg.handler.name.kerberosKeytabFile</code>	Optional (Required if <code>authType=Kerberos</code>)	Relative or absolute path to a Kerberos keytab file.	-	The keytab file allows the HDFS Handler to access a password to perform a kinit operation for Kerberos security.
<code>gg.handler.name.kerberosPrincipal</code>	Optional (Required if <code>authType=Kerberos</code>)	A legal Kerberos principal name like <code>user/FQDN@MY.REALM</code> .	-	The Kerberos principal name for Kerberos authentication.
<code>gg.handler.name.schemaFilePath</code>	Optional	-	<code>null</code>	Set to a legal path in HDFS so that schemas (if available) are written in that HDFS directory. Schemas are currently only available for Avro and JSON formatters. In the case of a metadata change event, the schema is overwritten to reflect the schema change.

Property	Optional / Required	Legal Values	Default	Explanation
<code>gg.handler.name.compressionType</code> Applicable to Sequence File Format only.	Optional	block none record	none	Hadoop Sequence File Compression Type. Applicable only if <code>gg.handler.name.format</code> is set to <code>sequencefile</code>
<code>gg.handler.name.compressionCodec</code> Applicable to Sequence File and writing to HDFS is Avro OCF formats only.	Optional	org.apache.hadoop.io.compress.DefaultCodec org.apache.hadoop.io.compress.BZip2Codec org.apache.hadoop.io.compress.SnappyCodec org.apache.hadoop.io.compress.GzipCodec	org.apache.hadoop.io.compress.DefaultCodec	Hadoop Sequence File Compression Codec. Applicable only if <code>gg.handler.name.format</code> is set to <code>sequencefile</code>
<code>gg.handler.name.compressionCodec</code>	Optional	null snappy bzip2 xz deflate	null	Avro OCF Formatter Compression Code. This configuration controls the selection of the compression library to be used for Avro OCF files. Snappy includes native binaries in the Snappy JAR file and performs a Java-native traversal when compressing or decompressing. Use of Snappy may introduce runtime issues and platform porting issues that you may not experience when working with Java. You may need to perform additional testing to ensure that Snappy works on all of your required platforms. Snappy is an open source library, so Oracle cannot guarantee its ability to operate on all of your required platforms.
<code>gg.handler.name.openNextFileAtRoll</code>	Optional	true false	false	Applicable only to the HDFS Handler that is not writing an Avro OCF or sequence file to support extract, load, transform (ELT) situations. When set to <code>true</code> , this property creates a new file immediately on the occurrence of a file roll. File rolls can be triggered by any one of the following: • Metadata change • File roll interval elapsed • Inactivity interval elapsed Data files are being loaded into HDFS and a monitor program is monitoring the write directories waiting to consume the data. The monitoring programs use the appearance of a new file as a trigger so that the previous file can be consumed by the consuming application.

Property	Optional / Required	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.hsync</code>	Optional	true false	false	Set to use an <code>hflush</code> call to ensure that data is transferred from the HDFS Handler to the HDFS cluster. When set to <code>false</code>, <code>hflush</code> is called on open HDFS write streams at transaction commit to ensure write durability. Setting <code>hsync</code> to <code>true</code> calls <code>hsync</code> instead of <code>hflush</code> at transaction commit. Using <code>hsync</code> ensures that data has moved to the HDFS cluster and that the data is written to disk. This provides a higher level of write durability though it adversely effects performance. Also, it does not make the write data immediately available to analytic tools. For most applications setting this property to <code>false</code> is appropriate.

10.2.9.3.3 Review a Sample Configuration

The following is a sample configuration for the HDFS Handler from the Java Adapter properties file:

```
gg.handlerlist=hdfs
gg.handler.hdfs.type=hdfs
gg.handler.hdfs.mode=tx
gg.handler.hdfs.includeTokens=false
gg.handler.hdfs.maxFileSize=1g
gg.handler.hdfs.pathMappingTemplate=/ogg/${fullyQualifiedTableName}
gg.handler.hdfs.fileRollInterval=0
gg.handler.hdfs.inactivityRollInterval=0
gg.handler.hdfs.partitionByTable=true
gg.handler.hdfs.rollOnMetadataChange=true
gg.handler.hdfs.authType=none
gg.handler.hdfs.format=delimitedtext
```

10.2.9.3.4 Performance Considerations

The HDFS Handler calls the HDFS flush method on the HDFS write stream to flush data to the HDFS data nodes at the end of each transaction in order to maintain write durability. This is an expensive call and performance can adversely affect, especially in the case of transactions of one or few operations that result in numerous HDFS flush calls.

Performance of the HDFS Handler can be greatly improved by batching multiple small transactions into a single larger transaction. If you require high performance, configure batching functionality for the Replicat process. For more information, see [Replicat Grouping](#).

The HDFS client libraries spawn threads for every HDFS file stream opened by the HDFS Handler. Therefore, the number of threads executing in the JVM grows proportionally to the number of HDFS file streams that are open. Performance of the HDFS Handler may degrade as more HDFS file streams are opened. Configuring the HDFS Handler to write to many HDFS files (due to many source replication tables or extensive use of partitioning) may result in degraded performance. If your use case requires writing to many tables, then Oracle recommends that you enable the roll on time or roll on inactivity features to close HDFS file streams. Closing an HDFS file stream causes the HDFS client threads to terminate, and the associated resources can be reclaimed by the JVM.

10.2.9.3.5 Security

The HDFS cluster can be secured using Kerberos authentication. The HDFS Handler can connect to Kerberos secured cluster. The HDFS core-site.xml should be in the handlers classpath with the `hadoop.security.authentication` property set to `kerberos` and the `hadoop.security.authorization` property set to `true`. Additionally, you must set the following properties in the HDFS Handler Java configuration file:

```
gg.handler.name.authType=kerberos
gg.handler.name.kerberosPrincipalName=legal Kerberos principal name
gg.handler.name.kerberosKeytabFile=path to a keytab file that contains the password for
the Kerberos principal so that the HDFS Handler can programmatically perform the
Kerberos kinit operations to obtain a Kerberos ticket
```

You may encounter the inability to decrypt the Kerberos password from the keytab file. This causes the Kerberos authentication to fall back to interactive mode which cannot work because it is being invoked programmatically. The cause of this problem is that the Java Cryptography Extension (JCE) is not installed in the Java Runtime Environment (JRE). Ensure that the JCE is loaded in the JRE, see <http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html>.

10.2.9.4 Writing in HDFS in Avro Object Container File Format

The HDFS Handler includes specialized functionality to write to HDFS in Avro Object Container File (OCF) format. This Avro OCF is part of the Avro specification and is detailed in the Avro documentation at:

<https://avro.apache.org/docs/current/spec.html#Object+Container+Files>

Avro OCF format may be a good choice because it:

- integrates with Apache Hive (Raw Avro written to HDFS is not supported by Hive.)
- provides good support for schema evolution.

Configure the following to enable writing to HDFS in Avro OCF format:

To write row data to HDFS in Avro OCF format, configure the `gg.handler.name.format=avro_row_ocf` property.

To write operation data to HDFS in Avro OCF format, configure the `gg.handler.name.format=avro_op_ocf` property.

The HDFS and Avro OCF integration includes functionality to create the corresponding tables in Hive and update the schema for metadata change events. The configuration section provides information on the properties to enable integration with Hive. The Oracle GoldenGate Hive integration accesses Hive using the JDBC interface, so the Hive JDBC server must be running to enable this integration.

10.2.9.5 Generating HDFS File Names Using Template Strings

The HDFS Handler can dynamically generate HDFS file names using a template string. The template string allows you to generate a combination of keywords that are dynamically resolved at runtime with static strings to provide you more control of generated HDFS file names. You can control the template file name using the

`gg.handler.name.fileNameMappingTemplate` configuration property. The default value for this parameters is:

`${fullyQualifiedTableName}_${groupName}_${currentTimestamp}.txt`

See [Template Keywords](#).

Following are examples of legal templates and the resolved strings:

Legal Template Replacement

`${schemaName}.${tableName}__${groupName}_${currentTimestamp}.txt`
TEST.TABLE1__HDFS001_2017-07-05_04-31-23.123.txt

`${fullyQualifiedTableName}--${currentTimestamp}.avro`
ORACLE.TEST.TABLE1-2017-07-05_04-31-23.123.avro

`${fullyQualifiedTableName}_${currentTimestamp[yyyy-MM-ddTHH-mm-ss.SSS]}.json`
ORACLE.TEST.TABLE1_2017-07-05T04-31-23.123.json

Be aware of these restrictions when generating HDFS file names using templates:

- Generated HDFS file names must be legal HDFS file names.
- Oracle strongly recommends that you use `${groupName}` as part of the HDFS file naming template when using coordinated apply and breaking down source table data to different Replicat threads. The group name provides uniqueness of generated HDFS names that `${currentTimestamp}` alone does not guarantee. HDFS file name collisions result in an abend of the Replicat process.

10.2.9.6 Metadata Change Events

Metadata change events are now handled in the HDFS Handler. The default behavior of the HDFS Handler is to roll the current relevant file in the event of a metadata change event. This behavior allows for the results of metadata changes to at least be separated into different files. File rolling on metadata change is configurable and can be turned off.

To support metadata change events, the process capturing changes in the source database must support both DDL changes and metadata in trail. Oracle GoldenGate does not support DDL replication for all database implementations. See the Oracle GoldenGate installation and configuration guide for the appropriate database to determine whether DDL replication is supported.

10.2.9.7 Partitioning

The partitioning functionality uses the template mapper functionality to resolve partitioning strings. The result is that you have more control in how to partition source trail data. Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 21c release, all the keywords that are supported by the templating functionality are supported in HDFS partitioning.

For more information, see [Template Keywords](#).

Precondition

To use the partitioning functionality, ensure that the data is partitioned by the table. You cannot set the following configuration:

`gg.handler.name.partitionByTable=false`

Path Configuration

Assume that the path mapping template is configured as follows:

```
gg.handler.hdfs.pathMappingTemplate=/ogg/${fullyQualifiedTableName}
```

At runtime the path resolves as follows for the source table `DBO.ORDERS`:

```
/ogg/DBO.ORDERS
```

Partitioning Configuration

Configure the HDFS partitioning as follows; any of the keywords that are legal for templating are now legal for partitioning:

```
gg.handler.name.partitioner.fully qualified table name=templating keywords  
and/or  
constants
```

Example 1: The partitioning for the `DBO.ORDERS` table is set to the following:

```
gg.handler.hdfs.partitioner.DBO.ORDERS=par_sales_region=$  
{columnValue[SALES_REGION]}
```

This example can result in the following breakdown of files in HDFS:

```
/ogg/DBO.ORDERS/par_sales_region=west/data files  
/ogg/DBO.ORDERS/par_sales_region=east/data files  
/ogg/DBO.ORDERS/par_sales_region=north/data files  
/ogg/DBO.ORDERS/par_sales_region=south/data files
```

Example 2: The partitioning for the `DBO.ORDERS` table is set to the following:

```
gg.handler.hdfs.partitioner.DBO.ORDERS=par_sales_region=$  
{columnValue[SALES_REGION]}/par_state=${columnValue[STATE]}
```

This example can result in the following breakdown of files in HDFS:

```
/ogg/DBO.ORDERS/par_sales_region=west/par_state=CA/data files  
/ogg/DBO.ORDERS/par_sales_region=east/par_state=FL/data files  
/ogg/DBO.ORDERS/par_sales_region=north/par_state=MN/data files  
/ogg/DBO.ORDERS/par_sales_region=south/par_state=TX/data files
```

Ensure to be extra vigilant while configuring HDFS partitioning. If you choose partitioning column values that have a very large range of data values, then it results in partitioning to a proportional number of output data files. The HDFS client spawns multiple threads to service each open HDFS write stream. Partitioning to very large numbers of HDFS files can result in resource exhaustion of memory and/or threads.

Note

Starting GG for DAA 21c, the Automated Hive integration has been removed with the changes to support templating in control partitioning.

10.2.9.8 HDFS Additional Considerations

The Oracle HDFS Handler requires certain HDFS client libraries to be resolved in its classpath as a prerequisite for streaming data to HDFS.

For a list of required client JAR files by version, see [HDFS Handler Client Dependencies](#). The HDFS client jars do not ship with the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) product. The HDFS Handler supports multiple versions of HDFS, and the HDFS client jars must be the same version as the HDFS version to which the HDFS Handler is connecting. The HDFS client jars are open source and are freely available to download from sites such as the Apache Hadoop site or the maven central repository.

In order to establish connectivity to HDFS, the `HDFS core-site.xml` file must be in the classpath of the HDFS Handler. If the `core-site.xml` file is not in the classpath, the HDFS client code defaults to a mode that attempts to write to the local file system. Writing to the local file system instead of HDFS can be advantageous for troubleshooting, building a point of contact (POC), or as a step in the process of building an HDFS integration.

Another common issue is that data streamed to HDFS using the HDFS Handler may not be immediately available to GG for DAA analytic tools, such as Hive. This behavior commonly occurs when the HDFS Handler is in possession of an open write stream to an HDFS file. HDFS writes in blocks of 128 MB by default. HDFS blocks under construction are not always visible to analytic tools. Additionally, inconsistencies between file sizes when using the `-ls`, `-cat`, and `-get` commands in the HDFS shell may occur. This is an anomaly of HDFS streaming and is discussed in the HDFS specification. This anomaly of HDFS leads to a potential 128 MB per file blind spot in analytic data. This may not be an issue if you have a steady stream of replication data and do not require low levels of latency for analytic data from HDFS. However, this may be a problem in some use cases because closing the HDFS write stream finalizes the block writing. Data is immediately visible to analytic tools, and file sizing metrics become consistent again. Therefore, the new file rolling feature in the HDFS Handler can be used to close HDFS writes streams, making all data visible.

Important

The file rolling solution may present its own problems. Extensive use of file rolling can result in many small files in HDFS. Many small files in HDFS may result in performance issues in analytic tools.

You may also notice the HDFS inconsistency problem in the following scenarios.

- The HDFS Handler process crashes.
- A forced shutdown is called on the HDFS Handler process.
- A network outage or other issue causes the HDFS Handler process to abend.

In each of these scenarios, it is possible for the HDFS Handler to end without explicitly closing the HDFS write stream and finalizing the writing block. HDFS in its internal process ultimately recognizes that the write stream has been broken, so HDFS finalizes the write block. In this

scenario, you may experience a short term delay before the HDFS process finalizes the write block.

10.2.9.9 Best Practices

It is considered a Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) best practice for the HDFS cluster to operate on dedicated servers called cluster nodes. Edge nodes are server machines that host the applications to stream data to and retrieve data from the HDFS cluster nodes. Because the HDFS cluster nodes and the edge nodes are different servers, the following benefits are seen:

- The HDFS cluster nodes do not compete for resources with the applications interfacing with the cluster.
- The requirements for the HDFS cluster nodes and edge nodes probably differ. This physical topology allows the appropriate hardware to be tailored to specific needs.

It is a best practice for the HDFS Handler to be installed and running on an edge node and streaming data to the HDFS cluster using network connection. The HDFS Handler can run on any machine that has network visibility to the HDFS cluster. The installation of the HDFS Handler on an edge node requires that the `core-site.xml` files, and the dependency jars are copied to the edge node so that the HDFS Handler can access them. The HDFS Handler can also run collocated on a HDFS cluster node if required.

10.2.9.10 Troubleshooting the HDFS Handler

Troubleshooting of the HDFS Handler begins with the contents for the Java `log4j` file. Follow the directions in the Java Logging Configuration to configure the runtime to correctly generate the Java `log4j` log file.

- [Java Classpath](#)
- [Java Boot Options](#)
- [HDFS Connection Properties](#)
- [Handler and Formatter Configuration](#)

10.2.9.10.1 Java Classpath

Problems with the Java classpath are common. The usual indication of a Java classpath problem is a `ClassNotFoundException` in the Java `log4j` log file. The Java `log4j` log file can be used to troubleshoot this issue. Setting the log level to `DEBUG` allows for logging of each of the jars referenced in the `gg.classpath` object to be logged to the log file. In this way, you can ensure that all of the required dependency jars are resolved by enabling `DEBUG` level logging and search the log file for messages, as in the following:

```
2015-09-21 10:05:10 DEBUG ConfigClassPath:74 - ...adding to classpath: url="file://ggwork/hadoop/hadoop-2.6.0/share/hadoop/common/lib/guava-11.0.2.jar"
```

10.2.9.10.2 Java Boot Options

When running HDFS replicat with JRE 11, `StackOverflowError` is thrown. You can fix this issue by editing the `bootoptions` property in the Java Adapter Properties file as follows:

```
jvm.bootoptions=-Djdk.lang.ProcessReaperUseDefaultStackSize=true
```


10.2.9.10.3 HDFS Connection Properties

The contents of the HDFS core-site.xml file (including default settings) are output to the Java log4j log file when the logging level is set to DEBUG or TRACE. This output shows the connection properties to HDFS. Search for the following in the Java log4j log file:

```
2015-09-21 10:05:11 DEBUG HDFSConfiguration:58 - Begin - HDFS configuration object
contents for connection troubleshooting.
```

If the fs.defaultFS property points to the local file system, then the core-site.xml file is not properly set in the gg.classpath property.

```
Key: [fs.defaultFS] Value: [file:///].
```

This shows to the fs.defaultFS property properly pointed at and HDFS host and port.

```
Key: [fs.defaultFS] Value: [hdfs://hdfshost:9000].
```

10.2.9.10.4 Handler and Formatter Configuration

The Java log4j log file contains information on the configuration state of the HDFS Handler and the selected formatter. This information is output at the INFO log level. The output resembles the following:

```
2015-09-21 10:05:11 INFO AvroRowFormatter:156 - **** Begin Avro Row Formatter -
Configuration Summary ****
Operation types are always included in the Avro formatter output.
  The key for insert operations is [I].
  The key for update operations is [U].
  The key for delete operations is [D].
  The key for truncate operations is [T].
Column type mapping has been configured to map source column types to an
appropriate corresponding Avro type.
Created Avro schemas will be output to the directory [./dirdef].
Created Avro schemas will be encoded using the [UTF-8] character set.
In the event of a primary key update, the Avro Formatter will ABEND.
Avro row messages will not be wrapped inside a generic Avro message.
No delimiter will be inserted after each generated Avro message.
**** End Avro Row Formatter - Configuration Summary ****
```

```
2015-09-21 10:05:11 INFO HDFSHandler:207 - **** Begin HDFS Handler -
Configuration Summary ****
Mode of operation is set to tx.
Data streamed to HDFS will be partitioned by table.
Tokens will be included in the output.
The HDFS root directory for writing is set to [/ogg].
The maximum HDFS file size has been set to 1073741824 bytes.
Rolling of HDFS files based on time is configured as off.
Rolling of HDFS files based on write inactivity is configured as off.
Rolling of HDFS files in the case of a metadata change event is enabled.
HDFS partitioning information:
  The HDFS partitioning object contains no partitioning information.
HDFS Handler Authentication type has been configured to use [none]
**** End HDFS Handler - Configuration Summary ****
```

10.2.9.11 HDFS Handler Client Dependencies

This appendix lists the HDFS client dependencies for Apache Hadoop. The hadoop-client-x.x.x.jar is not distributed with Apache Hadoop nor is it mandatory to be in the classpath.

The `hadoop-client-x.x.x.jar` is an empty maven project with the purpose of aggregating all of the Hadoop client dependencies.

Maven groupId: `org.apache.hadoop`

Maven artifactId: `hadoop-client`

Maven version: the HDFS version numbers listed for each section

- [Hadoop Client Dependencies](#)

10.2.9.11.1 Hadoop Client Dependencies

This section lists the Hadoop client dependencies for each HDFS version.

- [HDFS 3.3.0](#)
- [HDFS 3.2.0](#)
- [HDFS 3.1.4](#)
- [HDFS 3.0.3](#)
- [HDFS 2.9.2](#)
- [HDFS 2.8.5](#)
- [HDFS 2.7.7](#)
- [HDFS 2.6.0](#)
- [HDFS 2.5.2](#)
- [HDFS 2.4.1](#)
- [HDFS 2.3.0](#)
- [HDFS 2.2.0](#)

10.2.9.11.1.1 HDFS 3.3.0

accessors-smart-1.2.jar
animal-sniffer-annotations-1.17.jar
asm-5.0.4.jar
avro-1.7.7.jar
azure-keyvault-core-1.0.0.jar
azure-storage-7.0.0.jar
checker-qual-2.5.2.jar
commons-beanutils-1.9.4.jar
commons-cli-1.2.jar
commons-codec-1.11.jar
commons-collections-3.2.2.jar
commons-compress-1.19.jar
commons-configuration2-2.1.1.jar
commons-io-2.5.jar
commons-lang3-3.7.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.6.jar
commons-text-1.4.jar
curator-client-4.2.0.jar
curator-framework-4.2.0.jar
curator-recipes-4.2.0.jar
dnsjava-2.1.7.jar
failureaccess-1.0.jar

gson-2.2.4.jar
guava-27.0-jre.jar
hadoop-annotations-3.3.0.jar
hadoop-auth-3.3.0.jar
hadoop-azure-3.3.0.jar
hadoop-client-3.3.0.jar
hadoop-common-3.3.0.jar
hadoop-hdfs-client-3.3.0.jar
hadoop-mapreduce-client-common-3.3.0.jar
hadoop-mapreduce-client-core-3.3.0.jar
hadoop-mapreduce-client-jobclient-3.3.0.jar
hadoop-shaded-protobuf_3_7-1.0.0.jar
hadoop-yarn-api-3.3.0.jar
hadoop-yarn-client-3.3.0.jar
hadoop-yarn-common-3.3.0.jar
htrace-core4-4.1.0-incubating.jar
httpclient-4.5.6.jar
httpcore-4.4.10.jar
j2objc-annotations-1.1.jar
jackson-annotations-2.10.3.jar
jackson-core-2.6.0.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.10.3.jar
jackson-jaxrs-base-2.10.3.jar
jackson-jaxrs-json-provider-2.10.3.jar
jackson-mapper-asl-1.9.13.jar
jackson-module-jaxb-annotations-2.10.3.jar
jakarta.activation-api-1.2.1.jar
jakarta.xml.bind-api-2.3.2.jar
javax.activation-api-1.2.0.jar
javax.servlet-api-3.1.0.jar
jaxb-api-2.2.11.jar
jcip-annotations-1.0-1.jar
jersey-client-1.19.jar
jersey-core-1.19.jar
jersey-servlet-1.19.jar
jetty-client-9.4.20.v20190813.jar
jetty-http-9.4.20.v20190813.jar
jetty-io-9.4.20.v20190813.jar
jetty-security-9.4.20.v20190813.jar
jetty-servlet-9.4.20.v20190813.jar
jetty-util-9.4.20.v20190813.jar
jetty-util-ajax-9.4.20.v20190813.jar
jetty-webapp-9.4.20.v20190813.jar
jetty-xml-9.4.20.v20190813.jar
jline-3.9.0.jar
json-smart-2.3.jar
jsp-api-2.1.jar
jsr305-3.0.2.jar
jsr311-api-1.1.1.jar
kerb-admin-1.0.1.jar
kerb-client-1.0.1.jar
kerb-common-1.0.1.jar
kerb-core-1.0.1.jar
kerb-crypto-1.0.1.jar
kerb-identity-1.0.1.jar
kerb-server-1.0.1.jar
kerb-simplekdc-1.0.1.jar
kerb-util-1.0.1.jar
kerby-asn1-1.0.1.jar
kerby-config-1.0.1.jar
kerby-pkix-1.0.1.jar

kerby-util-1.0.1.jar
kerby-xdr-1.0.1.jar
listenablefuture-9999.0-empty-to-avoid-conflict-with-guava.jar
log4j-1.2.17.jar
nimbus-jose-jwt-7.9.jar
okhttp-2.7.5.jar
okio-1.6.0.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
re2j-1.1.jar
slf4j-api-1.7.25.jar
snappy-java-1.0.5.jar
stax2-api-3.1.4.jar
token-provider-1.0.1.jar
websocket-api-9.4.20.v20190813.jar
websocket-client-9.4.20.v20190813.jar
websocket-common-9.4.20.v20190813.jar
wildfly-openssl-1.0.7.Final.jar
woodstox-core-5.0.3.jar

10.2.9.11.1.2 HDFS 3.2.0

accessors-smart-1.2.jar
asm-5.0.4.jar
avro-1.7.7.jar
azure-keyvault-core-1.0.0.jar
azure-storage-7.0.0.jar
commons-beanutils-1.9.3.jar
commons-cli-1.2.jar
commons-codec-1.11.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration2-2.1.1.jar
commons-io-2.5.jar
commons-lang3-3.7.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.6.jar
commons-text-1.4.jar
curator-client-2.12.0.jar
curator-framework-2.12.0.jar
curator-recipes-2.12.0.jar
dnsjava-2.1.7.jar
gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-3.2.0.jar
hadoop-auth-3.2.0.jar
hadoop-azure-3.2.0.jar
hadoop-client-3.2.0.jar
hadoop-common-3.2.0.jar
hadoop-hdfs-client-3.2.0.jar
hadoop-mapreduce-client-common-3.2.0.jar
hadoop-mapreduce-client-core-3.2.0.jar
hadoop-mapreduce-client-jobclient-3.2.0.jar
hadoop-yarn-api-3.2.0.jar
hadoop-yarn-client-3.2.0.jar
hadoop-yarn-common-3.2.0.jar
htrace-core4-4.1.0-incubating.jar
httpclient-4.5.2.jar
httpcore-4.4.4.jar
jackson-annotations-2.9.5.jar
jackson-core-2.6.0.jar

jackson-core-asl-1.9.13.jar
jackson-databind-2.9.5.jar
jackson-jaxrs-base-2.9.5.jar
jackson-jaxrs-json-provider-2.9.5.jar
jackson-mapper-asl-1.9.13.jar
jackson-module-jaxb-annotations-2.9.5.jar
javax.servlet-api-3.1.0.jar
jaxb-api-2.2.11.jar
jcip-annotations-1.0-1.jar
jersey-client-1.19.jar
jersey-core-1.19.jar
jersey-servlet-1.19.jar
jetty-security-9.3.24.v20180605.jar
jetty-servlet-9.3.24.v20180605.jar
jetty-util-9.3.24.v20180605.jar
jetty-util-ajax-9.3.24.v20180605.jar
jetty-webapp-9.3.24.v20180605.jar
jetty-xml-9.3.24.v20180605.jar
json-smart-2.3.jar
jsp-api-2.1.jar
jsr305-3.0.0.jar
jsr311-api-1.1.1.jar
kerb-admin-1.0.1.jar
kerb-client-1.0.1.jar
kerb-common-1.0.1.jar
kerb-core-1.0.1.jar
kerb-crypto-1.0.1.jar
kerb-identity-1.0.1.jar
kerb-server-1.0.1.jar
kerb-simplekdc-1.0.1.jar
kerb-util-1.0.1.jar
kerby-asn1-1.0.1.jar
kerby-config-1.0.1.jar
kerby-pkix-1.0.1.jar
kerby-util-1.0.1.jar
kerby-xdr-1.0.1.jar
log4j-1.2.17.jar
nimbus-jose-jwt-4.41.1.jar
okhttp-2.7.5.jar
okio-1.6.0.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
re2j-1.1.jar
slf4j-api-1.7.25.jar
snappy-java-1.0.5.jar
stax2-api-3.1.4.jar
token-provider-1.0.1.jar
wildfly-openssl-1.0.4.Final.jar
woodstox-core-5.0.3.jar
xz-1.0.jar

10.2.9.11.1.3 HDFS 3.1.4

accessors-smart-1.2.jar
animal-sniffer-annotations-1.17.jar
asm-5.0.4.jar
avro-1.7.7.jar
azure-keyvault-core-1.0.0.jar
azure-storage-7.0.0.jar
checker-qual-2.5.2.jar
commons-beanutils-1.9.4.jar
commons-cli-1.2.jar

commons-codec-1.11.jar
commons-collections-3.2.2.jar
commons-compress-1.19.jar
commons-configuration2-2.1.1.jar
commons-io-2.5.jar
commons-lang-2.6.jar
commons-lang3-3.4.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.6.jar
curator-client-2.13.0.jar
curator-framework-2.13.0.jar
curator-recipes-2.13.0.jar
error_prone_annotations-2.2.0.jar
failureaccess-1.0.jar
gson-2.2.4.jar
guava-27.0-jre.jar
hadoop-annotations-3.1.4.jar
hadoop-auth-3.1.4.jar
hadoop-azure-3.1.4.jar
hadoop-client-3.1.4.jar
hadoop-common-3.1.4.jar
hadoop-hdfs-client-3.1.4.jar
hadoop-mapreduce-client-common-3.1.4.jar
hadoop-mapreduce-client-core-3.1.4.jar
hadoop-mapreduce-client-jobclient-3.1.4.jar
hadoop-yarn-api-3.1.4.jar
hadoop-yarn-client-3.1.4.jar
hadoop-yarn-common-3.1.4.jar
htrace-core4-4.1.0-incubating.jar
httpclient-4.5.2.jar
httpcore-4.4.4.jar
j2objc-annotations-1.1.jar
jackson-annotations-2.9.10.jar
jackson-core-2.9.10.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.9.10.4.jar
jackson-jaxrs-base-2.9.10.jar
jackson-jaxrs-json-provider-2.9.10.jar
jackson-mapper-asl-1.9.13.jar
jackson-module-jaxb-annotations-2.9.10.jar
javax.servlet-api-3.1.0.jar
jaxb-api-2.2.11.jar
jcip-annotations-1.0-1.jar
jersey-client-1.19.jar
jersey-core-1.19.jar
jersey-servlet-1.19.jar
jetty-security-9.4.20.v20190813.jar
jetty-servlet-9.4.20.v20190813.jar
jetty-util-9.4.20.v20190813.jar
jetty-util-ajax-9.4.20.v20190813.jar
jetty-webapp-9.4.20.v20190813.jar
jetty-xml-9.4.20.v20190813.jar
json-smart-2.3.jar
jsp-api-2.1.jar
jsr305-3.0.2.jar
jsr311-api-1.1.1.jar
kerb-admin-1.0.1.jar
kerb-client-1.0.1.jar
kerb-common-1.0.1.jar
kerb-core-1.0.1.jar
kerb-crypto-1.0.1.jar

kerb-identity-1.0.1.jar
kerb-server-1.0.1.jar
kerb-simplekdc-1.0.1.jar
kerb-util-1.0.1.jar
kerby-asn1-1.0.1.jar
kerby-config-1.0.1.jar
kerby-pkix-1.0.1.jar
kerby-util-1.0.1.jar
kerby-xdr-1.0.1.jar
listenablefuture-9999.0-empty-to-avoid-conflict-with-guava.jar
log4j-1.2.17.jar
nimbus-jose-jwt-7.9.jar
okhttp-2.7.5.jar
okio-1.6.0.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
re2j-1.1.jar
slf4j-api-1.7.25.jar
snappy-java-1.0.5.jar
stax2-api-3.1.4.jar
token-provider-1.0.1.jar
woodstox-core-5.0.3.jar

10.2.9.11.1.4 HDFS 3.0.3

accessors-smart-1.2.jar
asm-5.0.4.jar
avro-1.7.7.jar
azure-keyvault-core-0.8.0.jar
azure-storage-5.4.0.jar
commons-beanutils-1.9.3.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration2-2.1.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-lang3-3.4.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.6.jar
curator-client-2.12.0.jar
curator-framework-2.12.0.jar
curator-recipes-2.12.0.jar
gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-3.0.3.jar
hadoop-auth-3.0.3.jar
hadoop-azure-3.0.3.jar
hadoop-client-3.0.3.jar
hadoop-common-3.0.3.jar
hadoop-hdfs-client-3.0.3.jar
hadoop-mapreduce-client-common-3.0.3.jar
hadoop-mapreduce-client-core-3.0.3.jar
hadoop-mapreduce-client-jobclient-3.0.3.jar
hadoop-yarn-api-3.0.3.jar
hadoop-yarn-client-3.0.3.jar
hadoop-yarn-common-3.0.3.jar
htrace-core4-4.1.0-incubating.jar
httpclient-4.5.2.jar
httpcore-4.4.4.jar

jackson-annotations-2.7.8.jar
jackson-core-2.7.8.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.7.8.jar
jackson-jaxrs-base-2.7.8.jar
jackson-jaxrs-json-provider-2.7.8.jar
jackson-mapper-asl-1.9.13.jar
jackson-module-jaxb-annotations-2.7.8.jar
javax.servlet-api-3.1.0.jar
jaxb-api-2.2.11.jar
jcip-annotations-1.0-1.jar
jersey-client-1.19.jar
jersey-core-1.19.jar
jersey-servlet-1.19.jar
jetty-security-9.3.19.v20170502.jar
jetty-servlet-9.3.19.v20170502.jar
jetty-util-9.3.19.v20170502.jar
jetty-util-ajax-9.3.19.v20170502.jar
jetty-webapp-9.3.19.v20170502.jar
jetty-xml-9.3.19.v20170502.jar
json-smart-2.3.jar
jsp-api-2.1.jar
jsr305-3.0.0.jar
jsr311-api-1.1.1.jar
kerb-admin-1.0.1.jar
kerb-client-1.0.1.jar
kerb-common-1.0.1.jar
kerb-core-1.0.1.jar
kerb-crypto-1.0.1.jar
kerb-identity-1.0.1.jar
kerb-server-1.0.1.jar
kerb-simplekdc-1.0.1.jar
kerb-util-1.0.1.jar
kerby-asn1-1.0.1.jar
kerby-config-1.0.1.jar
kerby-pkix-1.0.1.jar
kerby-util-1.0.1.jar
kerby-xdr-1.0.1.jar
log4j-1.2.17.jar
nimbus-jose-jwt-4.41.1.jar
okhttp-2.7.5.jar
okio-1.6.0.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
re2j-1.1.jar
slf4j-api-1.7.25.jar
snappy-java-1.0.5.jar
stax2-api-3.1.4.jar
token-provider-1.0.1.jar
woodstox-core-5.0.3.jar
xz-1.0.jar

10.2.9.11.1.5 HDFS 2.9.2

accessors-smart-1.2.jar
activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
asm-5.0.4.jar
avro-1.7.7.jar

azure-keyvault-core-0.8.0.jar
azure-storage-5.4.0.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-lang3-3.4.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.7.1.jar
curator-framework-2.7.1.jar
curator-recipes-2.7.1.jar
ehcache-3.3.1.jar
geronimo-jcache_1.0_spec-1.0-alpha-1.jar
gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-2.9.2.jar
hadoop-auth-2.9.2.jar
hadoop-azure-2.9.2.jar
hadoop-client-2.9.2.jar
hadoop-common-2.9.2.jar
hadoop-hdfs-client-2.9.2.jar
hadoop-mapreduce-client-app-2.9.2.jar
hadoop-mapreduce-client-common-2.9.2.jar
hadoop-mapreduce-client-core-2.9.2.jar
hadoop-mapreduce-client-jobclient-2.9.2.jar
hadoop-mapreduce-client-shuffle-2.9.2.jar
hadoop-yarn-api-2.9.2.jar
hadoop-yarn-client-2.9.2.jar
hadoop-yarn-common-2.9.2.jar
hadoop-yarn-registry-2.9.2.jar
hadoop-yarn-server-common-2.9.2.jar
HikariCP-java7-2.4.12.jar
htrace-core4-4.1.0-incubating.jar
httpclient-4.5.2.jar
httpcore-4.4.4.jar
jackson-annotations-2.4.0.jar
jackson-core-2.7.8.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.4.0.jar
jackson-jaxrs-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jackson-xc-1.9.13.jar
jaxb-api-2.2.2.jar
jcip-annotations-1.0-1.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jetty-sslengine-6.1.26.jar
jetty-util-6.1.26.jar
json-smart-2.3.jar
jsp-api-2.1.jar
jsr305-3.0.0.jar
leveldbjni-all-1.8.jar
log4j-1.2.17.jar
mssql-jdbc-6.2.1.jre7.jar

netty-3.7.0.Final.jar
nimbus-jose-jwt-4.41.1.jar
okhttp-2.7.5.jar
okio-1.6.0.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
servlet-api-2.5.jar
slf4j-api-1.7.25.jar
slf4j-log4j12-1.7.25.jar
snappy-java-1.0.5.jar
stax2-api-3.1.4.jar
stax-api-1.0-2.jar
woodstox-core-5.0.3.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar

10.2.9.11.1.6 HDFS 2.8.5

accessors-smart-1.2.jar
activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
asm-5.0.4.jar
avro-1.7.4.jar
azure-storage-2.2.0.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-lang3-3.3.2.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.7.1.jar
curator-framework-2.7.1.jar
curator-recipes-2.7.1.jar
gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-2.8.5.jar
hadoop-auth-2.8.5.jar
hadoop-azure-2.8.5.jar
hadoop-client-2.8.5.jar
hadoop-common-2.8.5.jar
hadoop-hdfs-client-2.8.5.jar
hadoop-mapreduce-client-app-2.8.5.jar
hadoop-mapreduce-client-common-2.8.5.jar
hadoop-mapreduce-client-core-2.8.5.jar
hadoop-mapreduce-client-jobclient-2.8.5.jar
hadoop-mapreduce-client-shuffle-2.8.5.jar
hadoop-yarn-api-2.8.5.jar
hadoop-yarn-client-2.8.5.jar
hadoop-yarn-common-2.8.5.jar
hadoop-yarn-server-common-2.8.5.jar

htrace-core4-4.0.1-incubating.jar
httpclient-4.5.2.jar
httpcore-4.4.4.jar
jackson-core-2.2.3.jar
jackson-core-asl-1.9.13.jar
jackson-jaxrs-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jackson-xc-1.9.13.jar
jaxb-api-2.2.2.jar
jcip-annotations-1.0-1.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jetty-sslengine-6.1.26.jar
jetty-util-6.1.26.jar
json-smart-2.3.jar
jsp-api-2.1.jar
jsr305-3.0.0.jar
leveldbjni-all-1.8.jar
log4j-1.2.17.jar
netty-3.7.0.Final.jar
nimbus-jose-jwt-4.41.1.jar
okhttp-2.4.0.jar
okio-1.4.0.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
servlet-api-2.5.jar
slf4j-api-1.7.10.jar
slf4j-log4j12-1.7.10.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar

10.2.9.11.1.7 HDFS 2.7.7

HDFS 2.7.7 (HDFS 2.7.0 is effectively the same, simply substitute 2.7.0 on the libraries versioned as 2.7.7)

activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.4.jar
azure-storage-2.0.0.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-lang3-3.3.2.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.7.1.jar

curator-framework-2.7.1.jar
curator-recipes-2.7.1.jar
gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-2.7.7.jar
hadoop-auth-2.7.7.jar
hadoop-azure-2.7.7.jar
hadoop-client-2.7.7.jar
hadoop-common-2.7.7.jar
hadoop-hdfs-2.7.7.jar
hadoop-mapreduce-client-app-2.7.7.jar
hadoop-mapreduce-client-common-2.7.7.jar
hadoop-mapreduce-client-core-2.7.7.jar
hadoop-mapreduce-client-jobclient-2.7.7.jar
hadoop-mapreduce-client-shuffle-2.7.7.jar
hadoop-yarn-api-2.7.7.jar
hadoop-yarn-client-2.7.7.jar
hadoop-yarn-common-2.7.7.jar
hadoop-yarn-server-common-2.7.7.jar
htrace-core-3.1.0-incubating.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-2.2.3.jar
jackson-core-asl-1.9.13.jar
jackson-jaxrs-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jackson-xc-1.9.13.jar
jaxb-api-2.2.2.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jetty-sslengine-6.1.26.jar
jetty-util-6.1.26.jar
jsp-api-2.1.jar
jsr305-3.0.0.jar
leveldbjni-all-1.8.jar
log4j-1.2.17.jar
netty-3.6.2.Final.jar
netty-all-4.0.23.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
servlet-api-2.5.jar
slf4j-api-1.7.10.jar
slf4j-log4j12-1.7.10.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xercesImpl-2.9.1.jar
xml-apis-1.3.04.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar

10.2.9.11.1.8 HDFS 2.6.0

activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar

```
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.6.0.jar
curator-framework-2.6.0.jar
curator-recipes-2.6.0.jar
gson-2.2.4.jar
guava-11.0.2.jar
hadoop-annotations-2.6.0.jar
hadoop-auth-2.6.0.jar
hadoop-client-2.6.0.jar
hadoop-common-2.6.0.jar
hadoop-hdfs-2.6.0.jar
hadoop-mapreduce-client-app-2.6.0.jar
hadoop-mapreduce-client-common-2.6.0.jar
hadoop-mapreduce-client-core-2.6.0.jar
hadoop-mapreduce-client-jobclient-2.6.0.jar
hadoop-mapreduce-client-shuffle-2.6.0.jar
hadoop-yarn-api-2.6.0.jar
hadoop-yarn-client-2.6.0.jar
hadoop-yarn-common-2.6.0.jar
hadoop-yarn-server-common-2.6.0.jar
htrace-core-3.0.4.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.9.13.jar
jackson-jaxrs-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jackson-xc-1.9.13.jar
jaxb-api-2.2.2.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jetty-util-6.1.26.jar
jsr305-1.3.9.jar
leveldbjni-all-1.8.jar
log4j-1.2.17.jar
netty-3.6.2.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
servlet-api-2.5.jar
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xercesImpl-2.9.1.jar
xml-apis-1.3.04.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar
```

10.2.9.11.1.9 HDFS 2.5.2

HDFS 2.5.2 (HDFS 2.5.1 and 2.5.0 are effectively the same, simply substitute 2.5.1 or 2.5.0 on the libraries versioned as 2.5.2)

activation-1.1.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
guava-11.0.2.jar
hadoop-annotations-2.5.2.jar
hadoop-auth-2.5.2.jar
hadoop-client-2.5.2.jar
hadoop-common-2.5.2.jar
hadoop-hdfs-2.5.2.jar
hadoop-mapreduce-client-app-2.5.2.jar
hadoop-mapreduce-client-common-2.5.2.jar
hadoop-mapreduce-client-core-2.5.2.jar
hadoop-mapreduce-client-jobclient-2.5.2.jar
hadoop-mapreduce-client-shuffle-2.5.2.jar
hadoop-yarn-api-2.5.2.jar
hadoop-yarn-client-2.5.2.jar
hadoop-yarn-common-2.5.2.jar
hadoop-yarn-server-common-2.5.2.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.9.13.jar
jackson-jaxrs-1.9.13.jar
jackson-mapper-asl-1.9.13.jar
jackson-xc-1.9.13.jar
jaxb-api-2.2.2.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jetty-util-6.1.26.jar
jsr305-1.3.9.jar
leveldbjni-all-1.8.jar
log4j-1.2.17.jar
netty-3.6.2.Final.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
servlet-api-2.5.jar
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.6.jar

10.2.9.11.1.10 HDFS 2.4.1

HDFS 2.4.1 (HDFS 2.4.0 is effectively the same, simply substitute 2.4.0 on the libraries versioned as 2.4.1)

```
activation-1.1.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
guava-11.0.2.jar
hadoop-annotations-2.4.1.jar
hadoop-auth-2.4.1.jar
hadoop-client-2.4.1.jar
hadoop-hdfs-2.4.1.jar
hadoop-mapreduce-client-app-2.4.1.jar
hadoop-mapreduce-client-common-2.4.1.jar
hadoop-mapreduce-client-core-2.4.1.jar
hadoop-mapreduce-client-jobclient-2.4.1.jar
hadoop-mapreduce-client-shuffle-2.4.1.jar
hadoop-yarn-api-2.4.1.jar
hadoop-yarn-client-2.4.1.jar
hadoop-yarn-common-2.4.1.jar
hadoop-yarn-server-common-2.4.1.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.8.8.jar
jackson-mapper-asl-1.8.8.jar
jaxb-api-2.2.2.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jetty-util-6.1.26.jar
jsr305-1.3.9.jar
log4j-1.2.17.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
servlet-api-2.5.jar
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.5.jar
hadoop-common-2.4.1.jar
```

10.2.9.11.1.11 HDFS 2.3.0

```
activation-1.1.jar
avro-1.7.4.jar
```

commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
guava-11.0.2.jar
hadoop-annotations-2.3.0.jar
hadoop-auth-2.3.0.jar
hadoop-client-2.3.0.jar
hadoop-common-2.3.0.jar
hadoop-hdfs-2.3.0.jar
hadoop-mapreduce-client-app-2.3.0.jar
hadoop-mapreduce-client-common-2.3.0.jar
hadoop-mapreduce-client-core-2.3.0.jar
hadoop-mapreduce-client-jobclient-2.3.0.jar
hadoop-mapreduce-client-shuffle-2.3.0.jar
hadoop-yarn-api-2.3.0.jar
hadoop-yarn-client-2.3.0.jar
hadoop-yarn-common-2.3.0.jar
hadoop-yarn-server-common-2.3.0.jar
httpclient-4.2.5.jar
httpcore-4.2.4.jar
jackson-core-asl-1.8.8.jar
jackson-mapper-asl-1.8.8.jar
jaxb-api-2.2.2.jar
jersey-core-1.9.jar
jetty-util-6.1.26.jar
jsr305-1.3.9.jar
log4j-1.2.17.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
servlet-api-2.5.jar
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
snappy-java-1.0.4.1.jar
stax-api-1.0-2.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.5.jar

10.2.9.11.1.12 HDFS 2.2.0

activation-1.1.jar
aopalliance-1.0.jar
asm-3.1.jar
avro-1.7.4.jar
commons-beanutils-1.7.0.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar

commons-digester-1.8.jar
commons-httpclient-3.1.jar
commons-io-2.1.jar
commons-lang-2.5.jar
commons-logging-1.1.1.jar
commons-math-2.1.jar
commons-net-3.1.jar
gmbal-api-only-3.0.0-b023.jar
grizzly-framework-2.1.2.jar
grizzly-http-2.1.2.jar
grizzly-http-server-2.1.2.jar
grizzly-http-servlet-2.1.2.jar
grizzly-rcm-2.1.2.jar
guava-11.0.2.jar
guice-3.0.jar
hadoop-annotations-2.2.0.jar
hadoop-auth-2.2.0.jar
hadoop-client-2.2.0.jar
hadoop-common-2.2.0.jar
hadoop-hdfs-2.2.0.jar
hadoop-mapreduce-client-app-2.2.0.jar
hadoop-mapreduce-client-common-2.2.0.jar
hadoop-mapreduce-client-core-2.2.0.jar
hadoop-mapreduce-client-jobclient-2.2.0.jar
hadoop-mapreduce-client-shuffle-2.2.0.jar
hadoop-yarn-api-2.2.0.jar
hadoop-yarn-client-2.2.0.jar
hadoop-yarn-common-2.2.0.jar
hadoop-yarn-server-common-2.2.0.jar
jackson-core-asl-1.8.8.jar
jackson-jaxrs-1.8.3.jar
jackson-mapper-asl-1.8.8.jar
jackson-xc-1.8.3.jar
javax.inject-1.jar
javax.servlet-3.1.jar
javax.servlet-api-3.0.1.jar
jaxb-api-2.2.2.jar
jaxb-impl-2.2.3-1.jar
jersey-client-1.9.jar
jersey-core-1.9.jar
jersey-grizzly2-1.9.jar
jersey-guice-1.9.jar
jersey-json-1.9.jar
jersey-server-1.9.jar
jersey-test-framework-core-1.9.jar
jersey-test-framework-grizzly2-1.9.jar
jettison-1.1.jar
jetty-util-6.1.26.jar
jsr305-1.3.9.jar
log4j-1.2.17.jar
management-api-3.0.0-b012.jar
paranamer-2.3.jar
protobuf-java-2.5.0.jar
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
snappy-java-1.0.4.1.jar
stax-api-1.0.1.jar
xmlenc-0.52.jar
xz-1.0.jar
zookeeper-3.4.5.jar

10.2.10 Apache Kafka

The Kafka Handler is designed to stream change data from an Oracle GoldenGate trail to a Kafka topic.

This chapter describes how to use the Kafka Handler.

- [Overview](#)
- [Apache Kafka Connect Handler](#)
The Kafka Connect Handler is an extension of the standard Kafka messaging functionality.
- [Apache Kafka REST Proxy](#)
The Kafka REST Proxy Handler to stream messages to the Kafka REST Proxy distributed by Confluent.

10.2.10.1 Overview

The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) Kafka Handler streams change capture data from an Oracle GoldenGate trail to a Kafka topic. Additionally, the Kafka Handler provides functionality to publish messages to a separate schema topic. Schema publication for Avro and JSON is supported.

Apache Kafka is an open source, distributed, partitioned, and replicated messaging service, see <http://kafka.apache.org/>.

Kafka can be run as a single instance or as a cluster on multiple servers. Each Kafka server instance is called a broker. A Kafka topic is a category or feed name to which messages are published by the producers and retrieved by consumers.

In Kafka, when the topic name corresponds to the fully-qualified source table name, the Kafka Handler implements a Kafka producer. The Kafka producer writes serialized change data capture, from multiple source tables to either a single configured topic or separating source operations, to different Kafka topics.

- [Introduction to Oracle GoldenGate Kafka Replicat](#)
- [Architecture Overview](#)
- [Apache Kafka](#)
The Kafka Handler is designed to stream change data from an Oracle GoldenGate trail to a Kafka topic.

10.2.10.1.1 Introduction to Oracle GoldenGate Kafka Replicat

Oracle GoldenGate Kafka Replicat is an integral component of Oracle GoldenGate for Distributed Applications and Analytics (Oracle GoldenGate for DAA) package. It uses a modular handler architecture, enabling flexible, and reliable data replication from an Oracle GoldenGate trail to a various target system.

Both the Kafka Handler and the Kafka Connect Handler can be used for Oracle GoldenGate Kafka Replicat. These purpose-built integration components that stream committed database transactions from an Oracle GoldenGate trail to Kafka topics in a scalable and fault-tolerant manner. This architecture allows Oracle GoldenGate to act as a high-performance bridge between relational databases and event-driven data platforms.

Oracle GoldenGate Kafka Replicat is compatible with all Kafka-compliant streaming platforms, including Confluent Platform, Amazon MSK, Azure Event Hubs for Kafka, and Oracle

Streaming Service. Compatibility details are maintained in the official [Oracle GoldenGate certification matrix](#)

To support diverse integration patterns, the Oracle GoldenGate Kafka Replicat implements both the Apache Kafka Core Client API and the Kafka Connect API. It supports multiple serialization formats, JSON, Avro, XML, and DelimitedText when using the Core API, and JSON, Avro, and Protobuf converters when using Kafka Connect. This flexibility allows users to align with their existing messaging standards, schema registries, and consumer applications without re-engineering downstream systems.

Key technical features include:

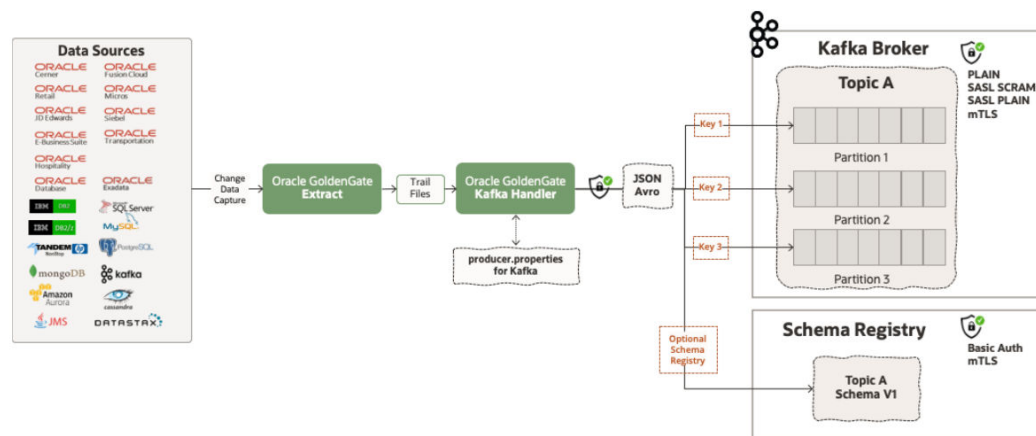
- **Seamless Data Streaming:** Efficiently transfers change data events from multiple source tables to designated Kafka topics using a single replication process.
- **Dynamic Topic Mapping:** Automated mapping of database operations to relevant Kafka topics, supporting row-level and table-level granularity.
- **Schema Registry:** Dynamic schema registration and management using Confluent Schema Registry, ensuring tight schema governance and easing schema evolution in streaming environments.

The following sections provide in-depth coverage of the Kafka replication architecture, deployment patterns, configuration best practices, as well as guidance for maximizing durability, performance, and data trustworthiness.

10.2.10.1.2 Architecture Overview

The Oracle GoldenGate for Kafka replication architecture is fundamentally an asynchronous, log-based CDC pipeline designed for high-throughput and transactional integrity. As illustrated in the diagram, the architecture bridges relational and non-relational source systems with the Apache Kafka ecosystem.

Figure 10-1 Apache Kafka Replication Architecture



The core design principle is decoupling. The use of the persistent trail files physically separates the Extract process (capture) from the Oracle GoldenGate Kafka Replicat (delivery). This is a vital architectural advantage because:

- **It enables backpressure tolerance:** If the Kafka Broker becomes temporarily unavailable or overloaded, the Oracle GoldenGate Kafka Replicat process can stop and resume seamlessly, while the Extract continues writing new changes to the trail files.

- **It minimizes latency risk:** The high-speed Extract process completes quickly, pushing database responsibility out of the critical path, while the Oracle GoldenGate Kafka Replicat handles the asynchronous network traffic and serialization overhead independently.

The overall architecture consists of the following major components:

- **Data Sources:** Oracle GoldenGate supports a wide array of transactional systems, including Oracle Databases (on-premises and cloud), Microsoft SQL Server, IBM DB2, MySQL, PostgreSQL, Cassandra, MongoDB, Amazon Aurora, JMS, and many more. This flexibility allows enterprises to centralize streaming ingestion from diverse operational systems. For the full list of supported source technologies, you can refer to Oracle GoldenGate certification matrix.
- **Change Data Capture (extract) and Trails:** As change data events are captured, Oracle GoldenGate standardizes all supported source changes into a unified, platform-independent Trail format. These Trails preserve transaction consistency and only emit committed events, enabling efficient, reliable, and consistent replication to downstream systems. Oracle GoldenGate provides a highly performant and feature-rich technology to detect change data events, especially from the Oracle Database. Oracle GoldenGate's kernel-level integration with Oracle Database allows for the broadest level of feature support and at the highest volumes while at the same time providing the least overhead of Oracle's CDC technologies. For more details about Oracle GoldenGate's CDC process, you can refer to Oracle GoldenGate Advantages.
- **Oracle GoldenGate Kafka Replicat:** The Oracle GoldenGate Kafka Replicat is a purpose-built pipeline to deliver committed change data into Apache Kafka in real-time. Oracle GoldenGate Kafka Replicat reads committed change data from GoldenGate trail files, then maps each transaction to one or Kafka Producer Record objects based on replicat configuration. Message keys can be dynamically assigned at runtime to support custom partitioning strategies within Kafka topics. The transformed messages are published to specified Kafka topics, supporting partitioning, ordering, and high-throughput requirements.

It supports different modes for packaging data: Transaction Mode (TX), which aggregates all operations in a transaction into a single Kafka message for consistency, and Operation Mode (OP), which generates an individual message for each change operation.

Oracle GoldenGate Kafka Replicat can serialize events using industry-standard formats such as Avro and JSON. When structured formats, such as Avro, and Protobuf are used, it supports Schema Propagation to a separate schema topic and can automatically register schemas into a Schema Registry. This functionality is critical for managing schema evolution and guaranteeing data integrity for downstream consumers.

- **Consumers and Downstream Systems:** Kafka consumers - whether analytical platforms, data-lake loaders, microservices, or CDC-driven applications - subscribe to topics to process change events in real-time.

In summary, this modular, decoupled Oracle GoldenGate Kafka Replicat architecture empowers organizations to establish real-time data integration pipelines that are scalable, extensible, and governed—laying a secure foundation for modern data-driven architectures.

10.2.10.1.3 Apache Kafka

The Kafka Handler is designed to stream change data from an Oracle GoldenGate trail to a Kafka topic.

This chapter describes how to use the Kafka Handler.

- [Detailed Functionality](#)
- [Setting Up and Running the Kafka Handler](#)

- [Schema Propagation](#)
- [Performance Considerations](#)
- [About Security](#)
- [Metadata Change Events](#)
- [Snappy Considerations](#)
- [Kafka Interceptor Support](#)

The Kafka Producer client framework supports the use of Producer Interceptors. A Producer Interceptor is simply a user exit from the Kafka Producer client whereby the Interceptor object is instantiated and receives notifications of Kafka message send calls and Kafka message send acknowledgement calls.
- [Kafka Partition Selection](#)

Kafka topics comprise one or more partitions. Distribution to multiple partitions is a good way to improve Kafka ingest performance, because the Kafka client parallelizes message sending to different topic/partition combinations. Partition selection is controlled by a following calculation in the Kafka client.
- [Troubleshooting](#)
- [Kafka Handler Client Dependencies](#)

What are the dependencies for the Kafka Handler to connect to Apache Kafka databases?

10.2.10.1.3.1 Detailed Functionality

Transaction Versus Operation Mode

The Kafka Handler sends instances of the Kafka ProducerRecord class to the Kafka producer API, which in turn publishes the ProducerRecord to a Kafka topic. The Kafka ProducerRecord effectively is the implementation of a Kafka message. The ProducerRecord has two components: a key and a value. Both the key and value are represented as byte arrays by the Kafka Handler. This section describes how the Kafka Handler publishes data.

Transaction Mode

The following configuration sets the Kafka Handler to transaction mode:

```
gg.handler.name.Mode=tx
```

In transaction mode, the serialized data is concatenated for every operation in a transaction from the source Oracle GoldenGate trail files. The contents of the concatenated operation data is the value of the Kafka ProducerRecord object. The key of the Kafka ProducerRecord object is NULL. The result is that Kafka messages comprise data from 1 to N operations, where N is the number of operations in the transaction.

For grouped transactions, all the data for all the operations are concatenated into a single Kafka message. Therefore, grouped transactions may result in very large Kafka messages that contain data for a large number of operations.

Operation Mode

The following configuration sets the Kafka Handler to operation mode:

```
gg.handler.name.Mode=op
```

In operation mode, the serialized data for each operation is placed into an individual ProducerRecord object as the value. The ProducerRecord key is the fully qualified table name of the source operation. The ProducerRecord is immediately sent using the Kafka Producer

API. This means that there is a 1 to 1 relationship between the incoming operations and the number of Kafka messages produced.

Topic Name Selection

The topic is resolved at runtime using this configuration parameter:

```
gg.handler.topicMappingTemplate
```

You can configure a static string, keywords, or a combination of static strings and keywords to dynamically resolve the topic name at runtime based on the context of the current operation, see [Using Templates to Resolve the Topic Name and Message Key](#).

Kafka Broker Settings

To configure topics to be created automatically, set the `auto.create.topics.enable` property to `true`. This is the default setting.

If you set the `auto.create.topics.enable` property to `false`, then you must manually create topics before you start the Replicat process.

Schema Propagation

The schema data for all tables is delivered to the schema topic that is configured with the `schemaTopicName` property. For more information, see [Schema Propagation](#).

10.2.10.1.3.2 Setting Up and Running the Kafka Handler

Instructions for configuring the Kafka Handler components and running the handler are described in this section.

You must install and correctly configure Kafka either as a single node or a clustered instance, see <http://kafka.apache.org/documentation.html>.

If you are using a Kafka distribution other than Apache Kafka, then consult the documentation for your Kafka distribution for installation and configuration instructions.

Zookeeper, a prerequisite component for Kafka and Kafka broker (or brokers), must be up and running.

Oracle recommends and considers it best practice that the data topic and the schema topic (if applicable) are preconfigured on the running Kafka brokers. You can create Kafka topics dynamically. However, this relies on the Kafka brokers being configured to allow dynamic topics.

If the Kafka broker is not colocated with the Kafka Handler process, then the remote host port must be reachable from the machine running the Kafka Handler.

- [Classpath Configuration](#)
- [Kafka Handler Configuration](#)
- [Java Adapter Properties File](#)
- [Kafka Producer Configuration File](#)
- [Using Templates to Resolve the Topic Name and Message Key](#)

The Kafka Handler provides functionality to resolve the topic name and the message key at runtime using a template configuration value. Templates allow you to configure static values and keywords. Keywords are used to dynamically resolve content at runtime and inject that resolved value into the resolved string.

- [Kafka Configuring with Kerberos on a Hadoop Platform](#)
- [Kafka SSL Support](#)
Kafka support SSL connectivity between Kafka clients and the Kafka cluster. SSL connectivity provides both authentication and encryption of messages transported between the client and the server.

10.2.10.1.3.2.1 Classpath Configuration

For the Kafka Handler to connect to Kafka and run, the Kafka Producer properties file and the Kafka client JARs must be configured in the `gg.classpath` configuration variable. The Kafka client JARs must match the version of Kafka that the Kafka Handler is connecting to. For a list of the required client JAR files by version, see [Kafka Handler Client Dependencies](#).

The recommended storage location for the Kafka Producer properties file is the Oracle GoldenGate `dirprm` directory.

The default location of the Kafka client JARs is `Kafka_Home/libs/*`.

The `gg.classpath` must be configured precisely. The path of the Kafka Producer Properties file must contain the path with no wildcard appended. If the `*` wildcard is included in the path to the Kafka Producer Properties file, the file is not picked up. Conversely, path to the dependency JARs must include the `*` wild card character in order to include all the JAR files in that directory in the associated classpath. Do *not* use `*.jar`. The following is an example of the correctly configured classpath:

```
gg.classpath={kafka install dir}/libs/*
```

10.2.10.1.3.2.2 Kafka Handler Configuration

The following are the configurable values for the Kafka Handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the Kafka Handler, you must first configure the handler type by specifying `gg.handler.name.type=kafka` and the other Kafka properties as follows:

Table 10-10 Configuration Properties for Kafka Handler

Property Name	Required / Optional	Property Value	Default	Description
<code>gg.handlerlist</code>	Required	<i>name</i> (choice of any name)	None	List of handlers to be used.
<code>gg.handler.name.type</code>	Required	kafka	None	Type of handler to use.
<code>gg.handler.name.KafkaProducerConfigFile</code>	Optional	Any custom file name	kafka-producer-default.properties	Filename in classpath that holds Apache Kafka properties to configure the Apache Kafka producer.
<code>gg.handler.name.Format</code>	Optional	Formatter class or short code.	delimitedtext	Formatter to use to format payload. Can be one of xml, delimitedtext, json, json_row, avro_row, avro_op
<code>gg.handler.name.SchemaTopicName</code>	Required when schema delivery is required.	Name of the schema topic.	None	Topic name where schema data will be delivered. If this property is not set, schema will not be propagated. Schemas will be propagated only for Avro formatters.

Table 10-10 (Cont.) Configuration Properties for Kafka Handler

Property Name	Required / Optional	Property Value	Default	Description
<code>gg.handler.name.SchemaPrClassName</code>	Optional	Fully qualified class name of a custom class that implements Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) Kafka Handler's <code>CreateProducerRecord</code> Java Interface.	Provided this implementation class: <code>oracle.goldengate.handler.kafka.ProducerRecord</code>	Schema is also propagated as a <code>ProducerRecord</code> . The default key is the fully qualified table name. If this needs to be changed for schema records, the custom implementation of the <code>CreateProducerRecord</code> interface needs to be created and this property needs to be set to point to the fully qualified name of the new class.
<code>gg.handler.name.mode</code>	Optional	<code>tx/op</code>	<code>tx</code>	With Kafka Handler operation mode, each change capture data record (Insert, Update, Delete, and so on) payload is represented as a Kafka Producer Record and is flushed one at a time. With Kafka Handler in transaction mode, all operations within a source transaction are represented as a single Kafka Producer record. This combined byte payload is flushed on a transaction Commit event.
<code>gg.handler.name.topicMappingTemplate</code>	Required	A template string value to resolve the Kafka topic name at runtime.	None	See Template Keywords .
<code>gg.handler.name.keyMappingTemplate</code>	Required	A template string value to resolve the Kafka message key at runtime.	None	See Template Keywords .
<code>gg.handler.name.metaHeadersTemplate</code>	Optional	Comma delimited list of metacolumn keywords.	None	Allows the user to select metacolumns to inject context-based key value pairs into Kafka message headers using the metacolumn keyword syntax.

10.2.10.1.3.2.3 Java Adapter Properties File

The following is a sample configuration for the Kafka Handler from the Adapter properties file:

```
gg.handlerlist = kafkahandler
gg.handler.kafkahandler.Type = kafka
gg.handler.kafkahandler.KafkaProducerConfigFile = custom_kafka_producer.properties
gg.handler.kafkahandler.topicMappingTemplate=oggtopic
gg.handler.kafkahandler.keyMappingTemplate=${currentTimestamp}
gg.handler.kafkahandler.Format = avro_op
gg.handler.kafkahandler.SchemaTopicName = oggSchemaTopic
gg.handler.kafkahandler.SchemaPrClassName = com.company.kafkaProdRec.SchemaRecord
gg.handler.kafkahandler.Mode = tx
```

You can find a sample Replicat configuration and a Java Adapter Properties file for a Kafka integration in the following directory:

GoldenGate_install_directory/AdapterExamples/big-data/kafka

10.2.10.1.3.2.4 Kafka Producer Configuration File

The Kafka Handler must access a Kafka producer configuration file in order to publish messages to Kafka. The file name of the Kafka producer configuration file is controlled by the following configuration in the Kafka Handler properties.

```
gg.handler.kafkahandler.KafkaProducerConfigFile=custom_kafka_producer.properties
```

The Kafka Handler attempts to locate and load the Kafka producer configuration file by using the Java classpath. Therefore, the Java classpath must include the directory containing the Kafka Producer Configuration File.

The Kafka producer configuration file contains Kafka proprietary properties. The Kafka documentation provides configuration information for the 0.8.2.0 Kafka producer interface properties. The Kafka Handler uses these properties to resolve the host and port of the Kafka brokers, and properties in the Kafka producer configuration file control the behavior of the interaction between the Kafka producer client and the Kafka brokers.

A sample of configuration file for the Kafka producer is as follows:

```
bootstrap.servers=localhost:9092
acks = 1
compression.type = gzip
reconnect.backoff.ms = 1000

value.serializer = org.apache.kafka.common.serialization.ByteArraySerializer
key.serializer = org.apache.kafka.common.serialization.ByteArraySerializer
# 100KB per partition
batch.size = 102400
linger.ms = 0
max.request.size = 1048576
send.buffer.bytes = 131072
```

- [Encrypt Kafka Producer Properties](#)

10.2.10.1.3.2.4.1 Encrypt Kafka Producer Properties

The sensitive properties within the Kafka Producer Configuration File can be encrypted using the Oracle GoldenGate Credential Store.

For more information about how to use Credential Store, see [Using Identities in Oracle GoldenGate Credential Store](#).

For example, the following kafka property:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule
required
username="alice" password="alice";
```

can be replaced with:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule
required
username=ORACLEWALLETUSERNAME[alias domain_name]
password=ORACLEWALLETPASSWORD[alias
domain_name];
```

10.2.10.1.3.2.5 Using Templates to Resolve the Topic Name and Message Key

The Kafka Handler provides functionality to resolve the topic name and the message key at runtime using a template configuration value. Templates allow you to configure static values

and keywords. Keywords are used to dynamically resolve content at runtime and inject that resolved value into the resolved string.

The templates use the following configuration properties:

```
gg.handler.name.topicMappingTemplate
gg.handler.name.keyMappingTemplate
```

Template Modes

Source database transactions are made up of one or more individual operations that are the individual inserts, updates, and deletes. The Kafka Handler can be configured to send one message per operation (insert, update, delete), or alternatively can be configured to group operations into messages at the transaction level. Many template keywords resolve data based on the context of an individual source database operation. Therefore, many of the keywords do *not* work when sending messages at the transaction level. For example, using `{fullyQualifiedTableName}` does not work when sending messages at the transaction level rather it resolves to the qualified source table name for an operation. However, transactions can contain multiple operations for many source tables. Resolving the fully qualified table name for messages at the transaction level is non-deterministic so abends at runtime.

For more information about the Template Keywords, see [Template Keywords](#).
See [Example Templates](#).

10.2.10.1.3.2.6 Kafka Configuring with Kerberos on a Hadoop Platform

Use these steps to configure a Kafka Handler Replicat with Kerberos to enable a Cloudera instance to process an Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) trail to a Kafka topic:

1. In GGSCI, add a Kafka Replicat:

```
GGSCI> add replicat kafka, exttrail dirdat/gg
```

2. Configure a prm file with these properties:

```
replicat kafka
discardfile ./dirrpt/kafkax.dsc, purge
SETENV (TZ=PST8PDT)
GETTRUNCATES
GETUPDATEBEFORES
ReportCount Every 1000 Records, Rate
MAP qasource.*, target qatarget.*;
```

3. Configure a Replicat properties file as follows:

```
###KAFKA Properties file ###
gg.log=log4j
gg.log.level=info
gg.report.time=30sec

###Kafka Classpath settings ###
gg.classpath=/opt/cloudera/parcels/KAFFKA-2.1.0-1.2.1.0.p0.115/lib/kafka/libs/*
jvm.bootoptions=-Xmx64m -Xms64m -Djava.class.path=./ggjava/ggjava.jar -
Dlog4j.configuration=log4j.properties -Djava.security.auth.login.config=/scratch/
ydama/ogg/v123211/dirprm/jaas.conf -Djava.security.krb5.conf=/etc/krb5.conf

### Kafka handler properties ###
gg.handlerlist = kafkahandler
gg.handler.kafkahandler.type=kafka
gg.handler.kafkahandler.KafkaProducerConfigFile=kafka-producer.properties
gg.handler.kafkahandler.format=delimitedtext
gg.handler.kafkahandler.format.PkUpdateHandling=update
```

```
gg.handler.kafkahandler.mode=op
gg.handler.kafkahandler.format.includeCurrentTimestamp=false
gg.handler.kafkahandler.format.fieldDelimiter=|
gg.handler.kafkahandler.format.lineDelimiter=CDATA[\n]
gg.handler.kafkahandler.topicMappingTemplate=myoggtopic
gg.handler.kafkahandler.keyMappingTemplate=${position}
```

4. Configure a Kafka Producer file with these properties:

```
bootstrap.servers=10.245.172.52:9092
acks=1
#compression.type=snappy
reconnect.backoff.ms=1000
value.serializer=org.apache.kafka.common.serialization.ByteArraySerializer
key.serializer=org.apache.kafka.common.serialization.ByteArraySerializer
batch.size=1024
linger.ms=2000

security.protocol=SASL_PLAINTEXT

sasl.kerberos.service.name=kafka
sasl.mechanism=GSSAPI
```

5. Configure a jaas.conf file with these properties:

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
storeKey=true
keyTab="/scratch/ydama/ogg/v123211/dirtmp/keytabs/slc06unm/kafka.keytab"
principal="kafka/slc06unm.us.oracle.com@HADOOPTTEST.Oracle.COM";
};
```

6. Ensure that you have the latest key.tab files from the Cloudera instance to connect secured Kafka topics.
7. Start the Replicat from GGSCI and make sure that it is running with INFO ALL.
8. Review the Replicat report to see the total number of records processed. The report is similar to:

```
Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA)

Copyright (c) 2007, 2018. Oracle and/or its affiliates. All rights reserved

Built with Java 1.8.0_161 (class version: 52.0)

2018-08-05 22:15:28 INFO OGG-01815 Virtual Memory Facilities for: COM
anon alloc: mmap(MAP_ANON) anon free: munmap
file alloc: mmap(MAP_SHARED) file free: munmap
target directories:
/scratch/ydama/ogg/v123211/dirtmp.

Database Version:

Database Language and Character Set:

*****
** Run Time Messages **
*****

2018-08-05 22:15:28 INFO OGG-02243 Opened trail file /scratch/ydama/ogg/v123211/
dirdat/kfkCustR/gg000000 at 2018-08-05 22:15:28.258810.
```

2018-08-05 22:15:28 INFO OGG-03506 The source database character set, as determined from the trail file, is UTF-8.

2018-08-05 22:15:28 INFO OGG-06506 Wildcard MAP resolved (entry qasource.*): MAP "QASOURCE"."BDCUSTMER1", target qatarget."BDCUSTMER1".

2018-08-05 22:15:28 INFO OGG-02756 The definition for table QASOURCE.BDCUSTMER1 is obtained from the trail file.

2018-08-05 22:15:28 INFO OGG-06511 Using following columns in default map by name: CUST_CODE, NAME, CITY, STATE.

2018-08-05 22:15:28 INFO OGG-06510 Using the following key columns for target table qatarget.BDCUSTMER1: CUST_CODE.

2018-08-05 22:15:29 INFO OGG-06506 Wildcard MAP resolved (entry qasource.*): MAP "QASOURCE"."BDCUSTORD1", target qatarget."BDCUSTORD1".

2018-08-05 22:15:29 INFO OGG-02756 The definition for table QASOURCE.BDCUSTORD1 is obtained from the trail file.

2018-08-05 22:15:29 INFO OGG-06511 Using following columns in default map by name: CUST_CODE, ORDER_DATE, PRODUCT_CODE, ORDER_ID, PRODUCT_PRICE, PRODUCT_AMOUNT, TRANSACTION_ID.

2018-08-05 22:15:29 INFO OGG-06510 Using the following key columns for target table qatarget.BDCUSTORD1: CUST_CODE, ORDER_DATE, PRODUCT_CODE, ORDER_ID.

2018-08-05 22:15:33 INFO OGG-01021 Command received from GGSCI: STATS.

2018-08-05 22:16:03 INFO OGG-01971 The previous message, 'INFO OGG-01021', repeated 1 times.

2018-08-05 22:43:27 INFO OGG-01021 Command received from GGSCI: STOP.

* ** Run Time Statistics ** *

Last record for the last committed transaction is the following:

Trail name : /scratch/ydama/ogg/v123211/dirdat/kfkCustR/gg000000
Hdr-Ind : E (x45) Partition : . (x0c)
UndoFlag : . (x00) BeforeAfter: A (x41)
RecLength : 0 (x0000) IO Time : 2015-08-14 12:02:20.022027
IOType : 100 (x64) OrigNode : 255 (xff)
TransInd : . (x03) FormatType : R (x52)
SyskeyLen : 0 (x00) Incomplete : . (x00)
AuditRBA : 78233 AuditPos : 23968384
Continued : N (x00) RecCount : 1 (x01)

2015-08-14 12:02:20.022027 GGSPurgedata Len 0 RBA 6473
TDR Index: 2

Reading /scratch/ydama/ogg/v123211/dirdat/kfkCustR/gg000000, current RBA 6556, 20 records, m_file_seqno = 0, m_file_rba = 6556

Report at 2018-08-05 22:43:27 (activity since 2018-08-05 22:15:28)

From Table QASOURCE.BDCUSTMER1 to qatarget.BDCUSTMER1:
inserts: 5

```
# updates: 1
# deletes: 0
# discards: 0
From Table QASOURCE.BDCUSTORD1 to qatarget.BDCUSTORD1:
# inserts: 5
# updates: 3
# deletes: 5
# truncates: 1
# discards: 0
```

9. Ensure that the secure Kafka topic is created:

```
/kafka/bin/kafka-topics.sh --zookeeper slc06unm:2181 --list
myoggtopic
```

10. Review the contents of the secure Kafka topic:

a. Create a consumer.properties file containing:

```
security.protocol=SASL_PLAINTEXT
sas.l.kerberos.service.name=kafka
```

b. Set this environment variable:

```
export KAFKA_OPTS="-Djava.security.auth.login.config="/scratch/ogg/v123211/
dirprm/jaas.conf"
```

c. Run the consumer utility to check the records:

```
/kafka/bin/kafka-console-consumer.sh --bootstrap-server sys06:9092 --topic
myoggtopic --new-consumer --consumer.config consumer.properties
```

10.2.10.1.3.2.7 Kafka SSL Support

Kafka support SSL connectivity between Kafka clients and the Kafka cluster. SSL connectivity provides both authentication and encryption of messages transported between the client and the server.

SSL can be configured for server authentication (client authenticates server) but is generally configured for mutual authentication (both client and server authenticate each other). In an SSL mutual authentication, each side of the connection retrieves a certificate from its keystore and passes it to the other side of the connection, which verifies the certificate against the certificate in its truststore.

When you set up SSL, see the [Kafka documentation](#) for more information about the specific Kafka version that you are running. The Kafka documentation also provides information on how to do the following:

- Set up the Kafka cluster for SSL
- Create self signed certificates in a keystore/truststore file
- Configure the Kafka clients for SSL

Oracle recommends you to implement the SSL connectivity using the Kafka producer and consumer command line utilities before attempting to use it with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA). The SSL connectivity should be confirmed between the machine hosting GG for DAA and the Kafka cluster. This action proves that SSL connectivity is correctly set up and working prior to introducing GG for DAA. The following is an example of Kafka producer configuration with SSL mutual authentication:

```
bootstrap.servers=localhost:9092
acks=1
value.serializer=org.apache.kafka.common.serialization.ByteArraySerializer
key.serializer=org.apache.kafka.common.serialization.ByteArraySerializer
security.protocol=SSL
ssl.keystore.location=/var/private/ssl/server.keystore.jks
```

```
ssl.keystore.password=test1234
ssl.key.password=test1234
ssl.truststore.location=/var/private/ssl/server.truststore.jks
ssl.truststore.password=test1234
```

10.2.10.1.3.3 Schema Propagation

The Kafka Handler provides the ability to publish schemas to a schema topic. Currently, the Avro Row and Operation formatters are the only formatters that are enabled for schema publishing. If the Kafka Handler `schemaTopicName` property is set, then the schema is published for the following events:

- The Avro schema for a specific table is published the first time an operation for that table is encountered.
- If the Kafka Handler receives a metadata change event, the schema is flushed. The regenerated Avro schema for a specific table is published the next time an operation for that table is encountered.
- If the Avro wrapping functionality is enabled, then the generic wrapper Avro schema is published the first time that any operation is encountered. To enable the generic wrapper, Avro schema functionality is enabled in the Avro formatter configuration, see [Avro Row Formatter](#) and [The Avro Operation Formatter](#).

The Kafka `ProducerRecord` value is the schema, and the key is the fully qualified table name.

Because Avro messages directly depend on an Avro schema, user of Avro over Kafka may encounter issues. Avro messages are not human readable because they are binary. To deserialize an Avro message, the receiver must first have the correct Avro schema, but because each table from the source database results in a separate Avro schema, this can be difficult. The receiver of a Kafka message cannot determine which Avro schema to use to deserialize individual messages when the source Oracle GoldenGate trail file includes operations from multiple tables. To solve this problem, you can wrap the specialized Avro messages in a generic Avro message wrapper. This generic Avro wrapper provides the fully-qualified table name, the hashcode of the schema string, and the wrapped Avro message. The receiver can use the fully-qualified table name and the hashcode of the schema string to resolve the associated schema of the wrapped message, and then use that schema to deserialize the wrapped message.

10.2.10.1.3.4 Performance Considerations

For the best performance, Oracle recommends that you send the Kafka Handler to operate in operation mode.

```
gg.handler.name.mode = op
```

Additionally, Oracle recommends that you set the `batch.size` and `linger.ms` values in the Kafka Producer properties file. These values are highly dependent upon the use case scenario. Typically, higher values result in better throughput, but latency is increased. Smaller values in these properties reduces latency but overall throughput decreases.

Use of the Replicat variable `GROUPTRANSOPS` also improves performance. The recommended setting is 10000.

If the serialized operations from the source trail file must be delivered in individual Kafka messages, then the Kafka Handler must be set to operation mode.

```
gg.handler.name.mode = op
```

10.2.10.1.3.5 About Security

Kafka version 0.9.0.0 introduced security through SSL/TLS and SASL (Kerberos). You can secure the Kafka Handler using one or both of the SSL/TLS and SASL security offerings. The Kafka producer client libraries provide an abstraction of security functionality from the integrations that use those libraries. The Kafka Handler is effectively abstracted from security functionality. Enabling security requires setting up security for the Kafka cluster, connecting machines, and then configuring the Kafka producer properties file with the required security properties. For detailed instructions about securing the Kafka cluster, see the Kafka documentation at

You may encounter the inability to decrypt the Kerberos password from the keytab file. This causes the Kerberos authentication to fall back to interactive mode which cannot work because it is being invoked programmatically. The cause of this problem is that the Java Cryptography Extension (JCE) is not installed in the Java Runtime Environment (JRE). Ensure that the JCE is loaded in the JRE, see <http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html>.

10.2.10.1.3.6 Metadata Change Events

Metadata change events are now handled in the Kafka Handler. This is relevant only if you have configured a schema topic and the formatter used supports schema propagation (currently Avro row and Avro Operation formatters). The next time an operation is encountered for a table for which the schema has changed, the updated schema is published to the schema topic.

To support metadata change events, the Oracle GoldenGate process capturing changes in the source database must support the Oracle GoldenGate metadata in trail feature, which was introduced in Oracle GoldenGate 12c (12.2).

10.2.10.1.3.7 Snappy Considerations

The Kafka Producer Configuration file supports the use of compression. One of the configurable options is Snappy, an open source compression and decompression (codec) library that provides better performance than other codec libraries. The Snappy JAR does not run on all platforms. Snappy may work on Linux systems though may or may not work on other UNIX and Windows implementations. If you want to use Snappy compression, test Snappy on all required systems before implementing compression using Snappy. If Snappy does not port to all required systems, then Oracle recommends using an alternate codec library.

10.2.10.1.3.8 Kafka Interceptor Support

The Kafka Producer client framework supports the use of Producer Interceptors. A Producer Interceptor is simply a user exit from the Kafka Producer client whereby the Interceptor object is instantiated and receives notifications of Kafka message send calls and Kafka message send acknowledgement calls.

The typical use case for Interceptors is monitoring. Kafka Producer Interceptors must conform to the interface `org.apache.kafka.clients.producer.ProducerInterceptor`. The Kafka Handler supports Producer Interceptor usage.

The requirements to using Interceptors in the Handlers are as follows:

- The Kafka Producer configuration property `"interceptor.classes"` must be configured with the class name of the Interceptor(s) to be invoked.

- In order to invoke the Interceptor(s), the jar files plus any dependency jars must be available to the JVM. Therefore, the jar files containing the Interceptor(s) plus any dependency jars must be added to the `gg.classpath` in the Handler configuration file. For more information, see [Kafka documentation](#).

10.2.10.1.3.9 Kafka Partition Selection

Kafka topics comprise one or more partitions. Distribution to multiple partitions is a good way to improve Kafka ingest performance, because the Kafka client parallelizes message sending to different topic/partition combinations. Partition selection is controlled by a following calculation in the Kafka client.

(Hash of the Kafka message key) modulus (the number of partitions) = selected partition number

The Kafka message key is selected by the following configuration value:

```
gg.handler.{your handler name}.keyMappingTemplate=
```

If this parameter is set to a value which generates a static key, all messages will go to the same partition. The following is example of static keys:

```
gg.handler.{your handler name}.keyMappingTemplate=StaticValue
```

If this parameter is set to a value which generates a key that changes infrequently, partition selection changes infrequently. In the following example the table name is used as the message key. Every operation for a specific source table will have the same key and thereby route to the same partition:

```
gg.handler.{your handler name}.keyMappingTemplate=${tableName}
```

A null Kafka message key distributes to the partitions on a round-robin basis. To do this, set the following:

```
gg.handler.{your handler name}.keyMappingTemplate=${null}
```

The recommended setting for configuration of the mapping key is the following:

```
gg.handler.{your handler name}.keyMappingTemplate=${primaryKeys}
```

This generates a Kafka message key that is the concatenated and delimited primary key values.

Operations for each row should have a unique primary key(s) thereby generating a unique Kafka message key for each row. Another important consideration is Kafka messages sent to different partitions are not guaranteed to be delivered to a Kafka consumer in the original order sent. This is part of the Kafka specification. Order is only maintained within a partition. Using primary keys as the Kafka message key means that operations for the same row, which have the same primary key(s), generate the same Kafka message key, and therefore are sent to the same Kafka partition. In this way, the order is maintained for operations for the same row.

At the DEBUG log level the Kafka message coordinates (topic, partition, and offset) are logged to the `.log` file for successfully sent messages.

10.2.10.1.3.10 Troubleshooting

- [Verify the Kafka Setup](#)
- [Classpath Issues](#)
- [Invalid Kafka Version](#)
- [Kafka Producer Properties File Not Found](#)
- [Kafka Connection Problem](#)

10.2.10.1.3.10.1 Verify the Kafka Setup

You can use the command line Kafka producer to write dummy data to a Kafka topic, and you can use a Kafka consumer to read this data from the Kafka topic. Use this method to verify the setup and read/write permissions to Kafka topics on disk, see <http://kafka.apache.org/documentation.html#quickstart>.

10.2.10.1.3.10.2 Classpath Issues

Java classpath problems are common. Such problems may include a `ClassNotFoundException` problem in the `log4j` log file or may be an error resolving the classpath because of a typographic error in the `gg.classpath` variable. The Kafka client libraries do *not* ship with the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) product. You must obtain the correct version of the Kafka client libraries and properly configure the `gg.classpath` property in the Java Adapter Properties file to correctly resolve the Java the Kafka client libraries as described in [Classpath Configuration](#).

10.2.10.1.3.10.3 Invalid Kafka Version

The Kafka Handler does *not* support Kafka versions 0.8.2.2 or older. If you run an unsupported version of Kafka, a runtime Java exception, `java.lang.NoSuchMethodError`, occurs. It implies that the `org.apache.kafka.clients.producer.KafkaProducer.flush()` method cannot be found. If you encounter this error, migrate to Kafka version 0.9.0.0 or later.

10.2.10.1.3.10.4 Kafka Producer Properties File Not Found

This problem typically results in the following exception:

```
ERROR 2015-11-11 11:49:08,482 [main] Error loading the kafka producer properties
```

Check the `gg.handler.kafkahandler.KafkaProducerConfigFile` configuration variable to ensure that the Kafka Producer Configuration file name is set correctly. Check the `gg.classpath` variable to verify that the classpath includes the path to the Kafka Producer properties file, and that the path to the properties file does not contain a `*` wildcard at the end.

10.2.10.1.3.10.5 Kafka Connection Problem

This problem occurs when the Kafka Handler is unable to connect to Kafka. You receive the following warnings:

```
WARN 2015-11-11 11:25:50,784 [kafka-producer-network-thread | producer-1] WARN  
(Selector.java:276) - Error in I/O with localhost/127.0.0.1  
java.net.ConnectException: Connection refused
```

The connection retry interval expires, and the Kafka Handler process abends. Ensure that the Kafka Broker is running and that the host and port provided in the Kafka Producer Properties file are correct. You can use network shell commands (such as `netstat -l`) on the machine hosting the Kafka broker to verify that Kafka is listening on the expected port.

10.2.10.1.3.11 Kafka Handler Client Dependencies

What are the dependencies for the Kafka Handler to connect to Apache Kafka databases?

The maven central repository artifacts for Kafka databases are:

Maven groupId: org.apache.kafka

Maven artifactId: kafka-clients

Maven version: the Kafka version numbers listed for each section

- [Kafka 2.8.0](#)
- [Kafka 2.7.0](#)
- [Kafka 2.6.0](#)
- [Kafka 2.5.1](#)
- [Kafka 2.4.1](#)
- [Kafka 2.3.1](#)

10.2.10.1.3.11.1 Kafka 2.8.0

kafka-clients-2.8.0.jar
lz4-java-1.7.1.jar
slf4j-api-1.7.30.jar
snappy-java-1.1.8.1.jar
zstd-jni-1.4.9-1.jar

10.2.10.1.3.11.2 Kafka 2.7.0

kafka-clients-2.7.0.jar
lz4-java-1.7.1.jar
slf4j-api-1.7.30.jar
snappy-java-1.1.7.7.jar
zstd-jni-1.4.5-6.jar

10.2.10.1.3.11.3 Kafka 2.6.0

kafka-clients-2.6.0.jar
lz4-java-1.7.1.jar
slf4j-api-1.7.30.jar
snappy-java-1.1.7.3.jar
zstd-jni-1.4.4-7.jar

10.2.10.1.3.11.4 Kafka 2.5.1

kafka-clients-2.5.1.jar
lz4-java-1.7.1.jar
slf4j-api-1.7.30.jar
snappy-java-1.1.7.3.jar
zstd-jni-1.4.4-7.jar

10.2.10.1.3.11.5 Kafka 2.4.1

kafka-clients-2.4.1.jar
lz4-java-1.6.0.jar
slf4j-api-1.7.28.jar
snappy-java-1.1.7.3.jar
zstd-jni-1.4.3-1.jar

10.2.10.1.3.11.6 Kafka 2.3.1

kafka-clients-2.3.1.jar
lz4-java-1.6.0.jar
slf4j-api-1.7.26.jar

snappy-java-1.1.7.3.jar
zstd-jni-1.4.0-1.jar

10.2.10.2 Apache Kafka Connect Handler

The Kafka Connect Handler is an extension of the standard Kafka messaging functionality.

This chapter describes how to use the Kafka Connect Handler.

- [Overview](#)
The Oracle GoldenGate Kafka Connect is an extension of the standard Kafka messaging functionality. Kafka Connect is a functional layer on top of the standard Kafka Producer and Consumer interfaces. It provides standardization for messaging to make it easier to add new source and target systems into your topology.
- [Detailed Functionality](#)
- [Setting Up and Running the Kafka Connect Handler](#)
- [Connecting to a Secure Schema Registry](#)
- [Kafka Connect Handler Performance Considerations](#)
- [Kafka Interceptor Support](#)
The Kafka Producer client framework supports the use of Producer Interceptors. A Producer Interceptor is simply a user exit from the Kafka Producer client whereby the Interceptor object is instantiated and receives notifications of Kafka message send calls and Kafka message send acknowledgement calls.
- [Kafka Partition Selection](#)
Kafka topics comprise one or more partitions. Distribution to multiple partitions is a good way to improve Kafka ingest performance, because the Kafka client parallelizes message sending to different topic/partition combinations. Partition selection is controlled by a following calculation in the Kafka client.
- [Troubleshooting the Kafka Connect Handler](#)
- [Kafka Connect Handler Client Dependencies](#)
What are the dependencies for the Kafka Connect Handler to connect to Apache Kafka Connect databases?

10.2.10.2.1 Overview

The Oracle GoldenGate Kafka Connect is an extension of the standard Kafka messaging functionality. Kafka Connect is a functional layer on top of the standard Kafka Producer and Consumer interfaces. It provides standardization for messaging to make it easier to add new source and target systems into your topology.

Confluent is a primary adopter of Kafka Connect and their Confluent Platform offering includes extensions over the standard Kafka Connect functionality. This includes Avro serialization and deserialization, and an Avro schema registry. Much of the Kafka Connect functionality is available in Apache Kafka. A number of open source Kafka Connect integrations are found at:

<https://www.confluent.io/product/connectors/>

The Kafka Connect Handler is a Kafka Connect source connector. You can capture database changes from any database supported by Oracle GoldenGate and stream that change of data through the Kafka Connect layer to Kafka. You can also connect to Oracle Event Hub Cloud Services (EHCS) with this handler.

Kafka Connect uses proprietary objects to define the schemas (`org.apache.kafka.connect.data.Schema`) and the messages

(`org.apache.kafka.connect.data.Struct`). The Kafka Connect Handler can be configured to manage what data is published and the structure of the published data.

The Kafka Connect Handler does *not* support any of the pluggable formatters that are supported by the Kafka Handler.

10.2.10.2.2 Detailed Functionality

The Kafka Connect framework provides converters to convert in-memory Kafka Connect messages to a serialized format suitable for transmission over a network. These converters are selected using configuration in the Kafka Producer properties file.

JSON Converter

Kafka Connect and the JSON converter is available as part of the Apache Kafka download. The JSON Converter converts the Kafka keys and values to JSONs which are then sent to a Kafka topic. You identify the JSON Converters with the following configuration in the Kafka Producer properties file:

```
key.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=true
```

The format of the messages is the message schema information followed by the payload information. JSON is a self describing format so you should not include the schema information in each message published to Kafka.

To omit the JSON schema information from the messages set the following:

```
key.converter.schemas.enable=false
value.converter.schemas.enable=false
```

Avro Converter

Confluent provides Kafka installations, support for Kafka, and extended functionality built on top of Kafka to help realize the full potential of Kafka. Confluent provides both open source versions of Kafka (Confluent Open Source) and an enterprise edition (Confluent Enterprise), which is available for purchase.

A common Kafka use case is to send Avro messages over Kafka. This can create a problem on the receiving end as there is a dependency for the Avro schema in order to deserialize an Avro message. Schema evolution can increase the problem because received messages must be matched up with the exact Avro schema used to generate the message on the producer side. Deserializing Avro messages with an incorrect Avro schema can cause runtime failure, incomplete data, or incorrect data. Confluent has solved this problem by using a schema registry and the Confluent schema converters.

The following shows the configuration of the Kafka Producer properties file.

```
key.converter=io.confluent.connect.avro.AvroConverter
value.converter=io.confluent.connect.avro.AvroConverter
key.converter.schema.registry.url=http://localhost:8081
value.converter.schema.registry.url=http://localhost:8081
```

When messages are published to Kafka, the Avro schema is registered and stored in the schema registry. When messages are consumed from Kafka, the exact Avro schema used to create the message can be retrieved from the schema registry to deserialize the Avro message. This creates matching of Avro messages to corresponding Avro schemas on the receiving side, which solves this problem.

Following are the requirements to use the Avro Converters:

- This functionality is available in both versions of Confluent Kafka (open source or enterprise).
- The Confluent schema registry service must be running.
- Source database tables must have an associated Avro schema. Messages associated with different Avro schemas must be sent to different Kafka topics.
- The Confluent Avro converters and the schema registry client must be available in the classpath.

The schema registry keeps track of Avro schemas by topic. Messages must be sent to a topic that has the same schema or evolving versions of the same schema. Source messages have Avro schemas based on the source database table schema so Avro schemas are unique for each source table. Publishing messages to a single topic for multiple source tables will appear to the schema registry that the schema is evolving every time the message sent from a source table that is different from the previous message.

Protobuf Converter

The Protobuf Converter allows Kafka Connect messages to be formatted as Google Protocol Buffers format. The Protobuf Converter integrates with the Confluent schema registry and this functionality is available in both the open source and enterprise versions of Confluent. Confluent added the Protobuf Converter starting in Confluent version 5.5.0.

The following shows the configuration to select the Protobuf Converter in the Kafka Producer Properties file:

```
key.converter=io.confluent.connect.protobuf.ProtobufConverter
value.converter=io.confluent.connect.protobuf.ProtobufConverter
key.converter.schema.registry.url=http://localhost:8081
value.converter.schema.registry.url=http://localhost:8081
```

The requirements to use the Protobuf Converter are as follows:

- This functionality is available in both versions of Confluent Kafka (open source or enterprise) starting in 5.5.0.
- The Confluent schema registry service must be running.
- Messages with different schemas (source tables) should be sent to different Kafka topics.
- The Confluent Protobuf converter and the schema registry client must be available in the classpath.

The schema registry keeps track of Protobuf schemas by topic. Messages must be sent to a topic that has the same schema or evolving versions of the same schema. Source messages have Protobuf schemas based on the source database table schema so Protobuf schemas are unique for each source table. Publishing messages to a single topic for multiple source tables will appear to the schema registry that the schema is evolving every time the message sent from a source table that is different from the previous message.

10.2.10.2.3 Setting Up and Running the Kafka Connect Handler

Instructions for configuring the Kafka Connect Handler components and running the handler are described in this section.

Classpath Configuration

Two things must be configured in the `gg.classpath` configuration variable so that the Kafka Connect Handler can connect to Kafka and run. The required items are the Kafka Producer properties file and the Kafka client JARs. The Kafka client JARs must match the version of Kafka that the Kafka Connect Handler is connecting to. For a listing of the required client JAR files by version, see [Kafka Handler Client Dependencies](#) [Kafka Connect Handler Client Dependencies](#). The recommended storage location for the Kafka Producer properties file is the Oracle GoldenGate `dirprm` directory.

The default location of the Kafka Connect client JARs is the `Kafka_Home/libs/*` directory.

The `gg.classpath` variable must be configured precisely. Pathing to the Kafka Producer properties file should contain the path with no wildcard appended. The inclusion of the asterisk (*) wildcard in the path to the Kafka Producer properties file causes it to be discarded. Pathing to the dependency JARs should include the * wildcard character to include all of the JAR files in that directory in the associated classpath. Do not use `*.jar`.

Following is an example of a correctly configured Apache Kafka classpath:

```
gg.classpath=dirprm:{kafka_install_dir}/libs/*
```

Following is an example of a correctly configured Confluent Kafka classpath:

```
gg.classpath={confluent_install_dir}/share/java/kafka-serde-tools/*:
{confluent_install_dir}/share/java/kafka/*:{confluent_install_dir}/share/java/confluent-
common/*
```

- [Kafka Connect Handler Configuration](#)
The automated output of meta-column fields in generated Kafka Connect messages has been removed as of Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 21c release.
- [Using Templates to Resolve the Topic Name and Message Key](#)
- [Configuring Security in the Kafka Connect Handler](#)

10.2.10.2.3.1 Kafka Connect Handler Configuration

The automated output of meta-column fields in generated Kafka Connect messages has been removed as of Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 21c release.

Meta-column fields can be configured as the following property:

```
gg.handler.name.metaColumnsTemplate
```

To output the metacolumns as in previous versions configure the following:

```
gg.handler.name.metaColumnsTemplate=${objectname[table]},${optype[op_type]},${
timestamp[op_ts]},${currenttimestamp[current_ts]},${position[pos]}
```

To also include the primary key columns and the tokens configure as follows:

```
gg.handler.name.metaColumnsTemplate=${objectname[table]},${optype[op_type]},$
{timestamp[op_ts]},${currenttimestamp[current_ts]},${position[pos]},$
{primarykeycolumns[primary_keys]},${alltokens[tokens]}
```

For more information see the configuration property:

```
gg.handler.name.metaColumnsTemplate
```

Table 10-11 Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.type</code>	Required	kafkaconnect	None	The configuration to select the Kafka Connect Handler.
<code>gg.handler.name.kafkaProducerConfi</code> <code>gFile</code>	Required	string	None	Name of the properties file containing the properties of the Kafka and Kafka Connect configuration properties. This file must be part of the classpath configured by the <code>gg.classpath</code> property.
<code>gg.handler.name.topicMappingTem</code> <code>plate</code>	Required	A template string value to resolve the Kafka topic name at runtime.	None	See Using Templates to Resolve the Topic Name and Message Key .
<code>gg.handler.name.keyMappingTem</code> <code>plate</code>	Required	A template string value to resolve the Kafka message key at runtime.	None	See Using Templates to Resolve the Topic Name and Message Key .
<code>gg.handler.name.includeTokens</code>	Optional	true false	false	Set to true to include a map field in output messages. The key is tokens and the value is a map where the keys and values are the token keys and values from the Oracle GoldenGate source trail file. Set to false to suppress this field.

Table 10-11 (Cont.) Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.messageFormatting</code>	Optional	<code>row op</code>	<code>row</code>	Controls how output messages are modeled. Selecting <code>row</code> and the output messages will be modeled as <code>row</code> . Set to <code>op</code> and the output messages will be modeled as operations messages.
<code>gg.handler.name</code> <code>.insertOpKey</code>	Optional	any string	<code>I</code>	The value of the field <code>op_type</code> to indicate an insert operation.
<code>gg.handler.name</code> <code>.updateOpKey</code>	Optional	any string	<code>U</code>	The value of the field <code>op_type</code> to indicate an insert operation.
<code>gg.handler.name</code> <code>.deleteOpKey</code>	Optional	any string	<code>D</code>	The value of the field <code>op_type</code> to indicate a delete operation.
<code>gg.handler.name</code> <code>.truncateOpKey</code>	Optional	any string	<code>T</code>	The value of the field <code>op_type</code> to indicate a truncate operation.
<code>gg.handler.name</code> <code>.treatAllColumnsAsStrings</code>	Optional	<code>true false</code>	<code>false</code>	Set to <code>true</code> to treat all output fields as strings. Set to <code>false</code> and the Handler will map the corresponding field type from the source trail file to the best corresponding Kafka Connect data type.
<code>gg.handler.name</code> <code>.mapLargeNumbersAsStrings</code>	Optional	<code>true false</code>	<code>false</code>	Large numbers are mapping to number fields as Doubles. It is possible to lose precision in certain scenarios. If set to <code>true</code> these fields will be mapped as Strings in order to preserve precision.

Table 10-11 (Cont.) Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.pkUpdateHandling</code>	Optional	<code>abend update delete-insert</code>	<code>abend</code>	Only applicable if modeling row messages <code>gg.handler.name.messageFormatting=row</code> . Not applicable if modeling operations messages as the before and after images are propagated to the message in the case of an update.
<code>gg.handler.name.metaColumnsTemplate</code>	Optional	Any of the metacolumns keywords.	None	A comma-delimited string consisting of one or more templated values that represent the template, see Metacolumn Keywords .
<code>gg.handler.name.includeIsMissingFields</code>	Optional	<code>true false</code>	<code>true</code>	Set to <code>true</code> to include an <code>extract{column_name}</code> . Set this property for each column to allow downstream applications to differentiate if a null value is actually null in the source trail file or if it is missing in the source trail file.
<code>gg.handler.name.enableDecimalLogicalType</code>	Optional	<code>true false</code>	<code>false</code>	Set to <code>true</code> to enable decimal logical types in Kafka Connect. Decimal logical types allow numbers which will not fit in a 64 bit data type to be represented.

Table 10-11 (Cont.) Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.oracleNumberScale</code>	Optional	Positive Integer	38	Only applicable if <code>gg.handler.name.enableDecimalLogicalType=true</code> . Some source data types do not have a fixed scale associated with them. Scale must be set for Kafka Connect decimal logical types. In the case of source types which do not have a scale in the metadata, the value of this parameter is used to set the scale.

Table 10-11 (Cont.) Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.EnableTimestampLogicalType</code>	Optional	<code>true false</code>	<code>false</code>	Set to <code>true</code> to enable the Kafka Connect timestamp logical type. The Kafka connect timestamp logical time is a integer measurement of milliseconds since the Java epoch. This means precision greater than milliseconds is not possible if the timestamp logical type is used. Use of this property requires that the <code>gg.format.timestamp</code> property be set. This property is the timestamp formatting string, which is used to determine the output of timestamps in string format. For example, <code>gg.format.timestamp=yyyy-MM-dd HH:mm:ss.SSS</code> . Ensure that the <code>goldengate.userexit.timestamp</code> property is not set in the configuration file. Setting this property prevents parsing the input timestamp into a Java object which is required for logical timestamps.

Table 10-11 (Cont.) Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.metaHeadersTemplate</code>	Optional	Comma delimited list of metacolumn keywords.	None	Allows the user to select metacolumns to inject context-based key value pairs into Kafka message headers using the metacolumn keyword syntax. See Metacolumn Keywords .
<code>gg.handler.name</code> <code>.schemaNamespace</code>	Optional	Any string without characters which violate the Kafka Connector Avro schema naming requirements.	None	Used to control the generated Kafka Connect schema name. If it is not set, then the schema name is the same as the qualified source table name. For example, if the source table is QASOURCE.TCUSTER, then the Kafka Connect schema name will be the same. This property allows you to control the generated schema name. For example, if this property is set to <code>com.example.company</code> , then the generated Kafka Connect schema name for the table QASOURCE.TCUSTER is <code>com.example.company.TCUSTER</code> .

Table 10-11 (Cont.) Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler.name .enableNonnulla ble	Optional	true false	false	The default behavior is to set all fields as nullable in the generated Kafka Connect schema. Set this parameter to <code>true</code> to honor the nullable value configured in the target metadata provided by the metadata provider. Setting this property to <code>true</code> can have some adverse side effects. 1. Setting a field to non-nullable means the field must have a value to be valid. If a field is set as non-nullable and the value is null or missing in the source trail file, a runtime error will result. 2. Setting a field to non-nullable means that truncate operations cannot be propagated. Truncate operations have no field values. The result is that the Kafka Connect converter serialization will field because there is no value for the field.

Table 10-11 (Cont.) Kafka Connect Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
				3. A schema change resulting in the addition of a non-nullable field will cause a schema backwards compatibility exception in the Confluent schema registry. If this occurs, users will need to adjust or disable the compatibility configuration of the Confluent schema registry.
<code>gg.handler.name</code> <code>.keyMetaColumns</code> Template	Optional	One or more of the metacolumn keywords separated by commas.	None	Resolves the message key using the metacolumns functionality and rendering the result as a structure. Use of this parameter overrides the use of <code>gg.handler.name</code> . <code>keyMappingTemplate</code> parameter that resolves the message key as a string using the mapping template functionality.

See [Using Templates to Resolve the Stream Name and Partition Name](#) for more information.

Review a Sample Configuration

```

gg.handlerlist=kafkaconnect
#The handler properties
gg.handler.kafkaconnect.type=kafkaconnect
gg.handler.kafkaconnect.kafkaProducerConfigFile=kafkaconnect.properties
gg.handler.kafkaconnect.mode=op
#The following selects the topic name based on the fully qualified table name
gg.handler.kafkaconnect.topicMappingTemplate=${fullyQualifiedTableName}
#The following selects the message key using the concatenated primary keys
gg.handler.kafkaconnect.keyMappingTemplate=${primaryKeys}
#The formatter properties

```

```
gg.handler.kafkaconnect.messageFormatting=row  
gg.handler.kafkaconnect.insertOpKey=I  
gg.handler.kafkaconnect.updateOpKey=U  
gg.handler.kafkaconnect.deleteOpKey=D  
gg.handler.kafkaconnect.truncateOpKey=T  
gg.handler.kafkaconnect.treatAllColumnsAsStrings=false  
gg.handler.kafkaconnect.pkUpdateHandling=abend
```

10.2.10.2.3.2 Using Templates to Resolve the Topic Name and Message Key

The Kafka Connect Handler provides functionality to resolve the topic name and the message key at runtime using a template configuration value. Templates allow you to configure static values and keywords. Keywords are used to dynamically replace the keyword with the context of the current processing. Templates are applicable to the following configuration parameters:

```
gg.handler.name.topicMappingTemplate  
gg.handler.name.keyMappingTemplate
```

Template Modes

The Kafka Connect Handler can only send operation messages. The Kafka Connect Handler cannot group operation messages into a larger transaction message.

For more information about the Template Keywords, see [Template Keywords](#).
For example templates, see [Example Templates](#).

10.2.10.2.3.3 Configuring Security in the Kafka Connect Handler

Kafka version 0.9.0.0 introduced security through SSL/TLS or Kerberos. The Kafka Connect Handler can be secured using SSL/TLS or Kerberos. The Kafka producer client libraries provide an abstraction of security functionality from the integrations utilizing those libraries. The Kafka Connect Handler is effectively abstracted from security functionality. Enabling security requires setting up security for the Kafka cluster, connecting machines, and then configuring the Kafka Producer properties file, that the Kafka Handler uses for processing, with the required security properties.

You may encounter the inability to decrypt the Kerberos password from the keytab file. This causes the Kerberos authentication to fall back to interactive mode which cannot work because it is being invoked programmatically. The cause of this problem is that the Java Cryptography Extension (JCE) is not installed in the Java Runtime Environment (JRE). Ensure that the JCE is loaded in the JRE, see <http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html>.

10.2.10.2.4 Connecting to a Secure Schema Registry

The customer topology for Kafka Connect may include a schema registry which is secured. This topic shows how to set the Kafka producer properties configured for connectivity to a secured schema registry.

SSL Mutual Auth

```
key.converter.schema.registry.ssl.truststore.location=  
key.converter.schema.registry.ssl.truststore.password=  
key.converter.schema.registry.ssl.keystore.location=  
key.converter.schema.registry.ssl.keystore.password=  
key.converter.schema.registry.ssl.key.password=  
value.converter.schema.registry.ssl.truststore.location=  
value.converter.schema.registry.ssl.truststore.password=  
value.converter.schema.registry.ssl.keystore.location=
```

```
value.converter.schema.registry.ssl.keystore.password=  
value.converter.schema.registry.ssl.key.password=
```

SSL Basic Auth

```
key.converter.basic.auth.credentials.source=USER_INFO  
key.converter.basic.auth.user.info=username:password  
key.converter.schema.registry.ssl.truststore.location=  
key.converter.schema.registry.ssl.truststore.password=  
value.converter.basic.auth.credentials.source=USER_INFO  
value.converter.basic.auth.user.info=username:password  
value.converter.schema.registry.ssl.truststore.location=  
value.converter.schema.registry.ssl.truststore.password=
```

Note

Security Recommendation: Oracle recommends storing credentials in Oracle GoldenGate Credential Store, Oracle Cloud Vault, or another enterprise credential management solution instead of storing plaintext passwords in configuration files.

10.2.10.2.5 Kafka Connect Handler Performance Considerations

There are multiple configuration settings both for the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) configuration and in the Kafka producer which affect performance.

The Oracle GoldenGate parameter have the greatest affect on performance is the Replicat GROUPTRANSOPS parameter. The GROUPTRANSOPS parameter allows Replicat to group multiple source transactions into a single target transaction. At transaction commit, the Kafka Connect Handler calls flush on the Kafka Producer to push the messages to Kafka for write durability followed by a checkpoint. The flush call is an expensive call and setting the Replicat GROUPTRANSOPS setting to larger amount allows the replicat to call the flush call less frequently thereby improving performance.

The default setting for GROUPTRANSOPS is 1000 and performance improvements can be obtained by increasing the value to 2500, 5000, or even 10000.

The Op mode `gg.handler.kafkaconnect.mode=op` parameter can also improve performance than the Tx mode `gg.handler.kafkaconnect.mode=tx`.

A number of Kafka Producer properties can affect performance. The following are the parameters with significant impact:

- `linger.ms`
- `batch.size`
- `acks`
- `buffer.memory`
- `compression.type`

Oracle recommends that you start with the default values for these parameters and perform performance testing to obtain a base line for performance. Review the Kafka documentation for each of these parameters to understand its role and adjust the parameters and perform additional performance testing to ascertain the performance effect of each parameter.

10.2.10.2.6 Kafka Interceptor Support

The Kafka Producer client framework supports the use of Producer Interceptors. A Producer Interceptor is simply a user exit from the Kafka Producer client whereby the Interceptor object is instantiated and receives notifications of Kafka message send calls and Kafka message send acknowledgement calls.

The typical use case for Interceptors is monitoring. Kafka Producer Interceptors must conform to the interface `org.apache.kafka.clients.producer.ProducerInterceptor`. The Kafka Connect Handler supports Producer Interceptor usage.

The requirements to using Interceptors in the Handlers are as follows:

- The Kafka Producer configuration property "interceptor.classes" must be configured with the class name of the Interceptor(s) to be invoked.
- In order to invoke the Interceptor(s), the jar files plus any dependency jars must be available to the JVM. Therefore, the jar files containing the Interceptor(s) plus any dependency jars must be added to the `gg.classpath` in the Handler configuration file. For more information, see [Kafka documentation](#).

10.2.10.2.7 Kafka Partition Selection

Kafka topics comprise one or more partitions. Distribution to multiple partitions is a good way to improve Kafka ingest performance, because the Kafka client parallelizes message sending to different topic/partition combinations. Partition selection is controlled by a following calculation in the Kafka client.

$(\text{Hash of the Kafka message key}) \bmod (\text{the number of partitions}) = \text{selected partition number}$

The Kafka message key is selected by the following configuration value:

```
gg.handler.{your handler name}.keyMappingTemplate=
```

If this parameter is set to a value which generates a static key, all messages will go to the same partition. The following is example of static keys:

```
gg.handler.{your handler name}.keyMappingTemplate=StaticValue
```

If this parameter is set to a value which generates a key that changes infrequently, partition selection changes infrequently. In the following example the table name is used as the message key. Every operation for a specific source table will have the same key and thereby route to the same partition:

```
gg.handler.{your handler name}.keyMappingTemplate=${tableName}
```

A null Kafka message key distributes to the partitions on a round-robin basis. To do this, set the following:

```
gg.handler.{your handler name}.keyMappingTemplate=${null}
```

The recommended setting for configuration of the mapping key is the following:

```
gg.handler.{your handler name}.keyMappingTemplate=${primaryKeys}
```

This generates a Kafka message key that is the concatenated and delimited primary key values.

Operations for each row should have a unique primary key(s) thereby generating a unique Kafka message key for each row. Another important consideration is Kafka messages sent to different partitions are not guaranteed to be delivered to a Kafka consumer in the original order sent. This is part of the Kafka specification. Order is only maintained within a partition. Using primary keys as the Kafka message key means that operations for the same row, which have the same primary key(s), generate the same Kafka message key, and therefore are sent to the same Kafka partition. In this way, the order is maintained for operations for the same row.

At the DEBUG log level the Kafka message coordinates (topic, partition, and offset) are logged to the `.log` file for successfully sent messages.

10.2.10.2.8 Troubleshooting the Kafka Connect Handler

- [Java Classpath for Kafka Connect Handler](#)
- [Invalid Kafka Version](#)
- [Kafka Producer Properties File Not Found](#)
- [Kafka Connection Problem](#)

10.2.10.2.8.1 Java Classpath for Kafka Connect Handler

Issues with the Java classpath are one of the most common problems. The indication of a classpath problem is a `ClassNotFoundException` in the Oracle GoldenGate Java `log4j` log file or an error while resolving the classpath if there is a typographic error in the `gg.classpath` variable.

The Kafka client libraries do not ship with the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) product. You are required to obtain the correct version of the Kafka client libraries and to properly configure the `gg.classpath` property in the Java Adapter Properties file to correctly resolve the Java the Kafka client libraries as described in [Setting Up and Running the Kafka Connect Handler](#).

10.2.10.2.8.2 Invalid Kafka Version

Kafka Connect was introduced in Kafka 0.9.0.0 version. The Kafka Connect Handler does not work with Kafka versions 0.8.2.2 and older. Attempting to use Kafka Connect with Kafka 0.8.2.2 version typically results in a `ClassNotFoundException` error at runtime.

10.2.10.2.8.3 Kafka Producer Properties File Not Found

Typically, the following exception message occurs:

```
ERROR 2015-11-11 11:49:08,482 [main] Error loading the kafka producer
properties
```

Verify that the `gg.handler.kafkahandler.KafkaProducerConfigFile` configuration property for the Kafka Producer Configuration file name is set correctly.

Ensure that the `gg.classpath` variable includes the path to the Kafka Producer properties file and that the path to the properties file does not contain a `*` wildcard at the end.

10.2.10.2.8.4 Kafka Connection Problem

Typically, the following exception message appears:

```
WARN 2015-11-11 11:25:50,784 [kafka-producer-network-thread | producer-1]
```

```
WARN (Selector.java:276) - Error in I/O with localhost/127.0.0.1  
java.net.ConnectException: Connection refused
```

When this occurs, the connection retry interval expires and the Kafka Connection Handler process abends. Ensure that the Kafka Brokers are running and that the host and port provided in the Kafka Producer properties file is correct.

Network shell commands (such as, `netstat -l`) can be used on the machine hosting the Kafka broker to verify that Kafka is listening on the expected port.

10.2.10.2.9 Kafka Connect Handler Client Dependencies

What are the dependencies for the Kafka Connect Handler to connect to Apache Kafka Connect databases?

The maven central repository artifacts for Kafka Connect databases are:

Maven groupId: org.apache.kafka

Maven artifactId: kafka-clients & connect-json

Maven version: the Kafka Connect version numbers listed for each section

- [Kafka 2.8.0](#)
- [Kafka 2.7.1](#)
- [Kafka 2.6.0](#)
- [Kafka 2.5.1](#)
- [Kafka 2.4.1](#)
- [Kafka 2.3.1](#)
- [Kafka 2.2.1](#)
- [Kafka 2.1.1](#)
- [Kafka 2.0.1](#)
- [Kafka 1.1.1](#)
- [Kafka 1.0.2](#)
- [Kafka 0.11.0.0](#)
- [Kafka 0.10.2.0](#)
- [Kafka 0.10.2.0](#)
- [Kafka 0.10.0.0](#)
- [Kafka 0.9.0.1](#)

10.2.10.2.9.1 Kafka 2.8.0

```
connect-api-2.8.0.jar  
connect-json-2.8.0.jar  
jackson-annotations-2.10.5.jar  
jackson-core-2.10.5.jar  
jackson-databind-2.10.5.1.jar  
jackson-datatype-jdk8-2.10.5.jar  
javax.ws.rs-api-2.1.1.jar  
kafka-clients-2.8.0.jar
```

```
lz4-java-1.7.1.jar  
slf4j-api-1.7.30.jar  
snappy-java-1.1.8.1.jar  
zstd-jni-1.4.9-1.jar
```

10.2.10.2.9.2 Kafka 2.7.1

```
connect-api-2.7.1.jar  
connect-json-2.7.1.jar  
jackson-annotations-2.10.5.jar  
jackson-core-2.10.5.jar  
jackson-databind-2.10.5.1.jar  
jackson-datatype-jdk8-2.10.5.jar  
javax.ws.rs-api-2.1.1.jar  
kafka-clients-2.7.1.jar  
lz4-java-1.7.1.jar  
slf4j-api-1.7.30.jar  
snappy-java-1.1.7.7.jar  
zstd-jni-1.4.5-6.jar
```

10.2.10.2.9.3 Kafka 2.6.0

```
connect-api-2.6.0.jar  
connect-json-2.6.0.jar  
jackson-annotations-2.10.2.jar  
jackson-core-2.10.2.jar  
jackson-databind-2.10.2.jar  
jackson-datatype-jdk8-2.10.2.jar  
javax.ws.rs-api-2.1.1.jar  
kafka-clients-2.6.0.jar  
lz4-java-1.7.1.jar  
slf4j-api-1.7.30.jar  
snappy-java-1.1.7.3.jar  
zstd-jni-1.4.4-7.jar
```

10.2.10.2.9.4 Kafka 2.5.1

```
connect-api-2.5.1.jar  
connect-json-2.5.1.jar  
jackson-annotations-2.10.2.jar  
jackson-core-2.10.2.jar  
jackson-databind-2.10.2.jar  
jackson-datatype-jdk8-2.10.2.jar  
javax.ws.rs-api-2.1.1.jar  
kafka-clients-2.5.1.jar  
lz4-java-1.7.1.jar  
slf4j-api-1.7.30.jar  
snappy-java-1.1.7.3.jar  
zstd-jni-1.4.4-7.jar
```

10.2.10.2.9.5 Kafka 2.4.1

```
kafka-clients-2.4.1.jar  
lz4-java-1.6.0.jar  
slf4j-api-1.7.28.jar  
snappy-java-1.1.7.3.jar  
zstd-jni-1.4.3-1.jar
```

10.2.10.2.9.6 Kafka 2.3.1

```
connect-api-2.3.1.jar  
connect-json-2.3.1.jar
```

jackson-annotations-2.10.0.jar
jackson-core-2.10.0.jar
jackson-databind-2.10.0.jar
jackson-datatype-jdk8-2.10.0.jar
javax.ws.rs-api-2.1.1.jar
kafka-clients-2.3.1.jar
lz4-java-1.6.0.jar
slf4j-api-1.7.26.jar
snappy-java-1.1.7.3.jar
zstd-jni-1.4.0-1.jar

10.2.10.2.9.7 Kafka 2.2.1

kafka-clients-2.2.1.jar
lz4-java-1.5.0.jar
slf4j-api-1.7.25.jar
snappy-java-1.1.7.2.jar
zstd-jni-1.3.8-1.jar

10.2.10.2.9.8 Kafka 2.1.1

audience-annotations-0.5.0.jar
connect-api-2.1.1.jar
connect-json-2.1.1.jar
jackson-annotations-2.9.0.jar
jackson-core-2.9.8.jar
jackson-databind-2.9.8.jar
javax.ws.rs-api-2.1.1.jar
jopt-simple-5.0.4.jar
kafka_2.12-2.1.1.jar
kafka-clients-2.1.1.jar
lz4-java-1.5.0.jar
metrics-core-2.2.0.jar
scala-library-2.12.7.jar
scala-logging_2.12-3.9.0.jar
scala-reflect-2.12.7.jar
slf4j-api-1.7.25.jar
snappy-java-1.1.7.2.jar
zkclient-0.11.jar
zookeeper-3.4.13.jar
zstd-jni-1.3.7-1.jar

10.2.10.2.9.9 Kafka 2.0.1

audience-annotations-0.5.0.jar
connect-api-2.0.1.jar
connect-json-2.0.1.jar
jackson-annotations-2.9.0.jar
jackson-core-2.9.7.jar
jackson-databind-2.9.7.jar
javax.ws.rs-api-2.1.jar
jopt-simple-5.0.4.jar
kafka_2.12-2.0.1.jar
kafka-clients-2.0.1.jar
lz4-java-1.4.1.jar
metrics-core-2.2.0.jar
scala-library-2.12.6.jar
scala-logging_2.12-3.9.0.jar
scala-reflect-2.12.6.jar
slf4j-api-1.7.25.jar
snappy-java-1.1.7.1.jar

zkclient-0.10.jar
zookeeper-3.4.13.jar

10.2.10.2.9.10 Kafka 1.1.1

kafka-clients-1.1.1.jar
lz4-java-1.4.1.jar
slf4j-api-1.7.25.jar
snappy-java-1.1.7.1.jar

10.2.10.2.9.11 Kafka 1.0.2

kafka-clients-1.0.2.jar
lz4-java-1.4.jar
slf4j-api-1.7.25.jar
snappy-java-1.1.4.jar

10.2.10.2.9.12 Kafka 0.11.0.0

connect-api-0.11.0.0.jar
connect-json-0.11.0.0.jar
jackson-annotations-2.8.0.jar
jackson-core-2.8.5.jar
jackson-databind-2.8.5.jar
jopt-simple-5.0.3.jar
kafka_2.11-0.11.0.0.jar
kafka-clients-0.11.0.0.jar
log4j-1.2.17.jar
lz4-1.3.0.jar
metrics-core-2.2.0.jar
scala-library-2.11.11.jar
scala-parser-combinators_2.11-1.0.4.jar
slf4j-api-1.7.25.jar
slf4j-log4j12-1.7.25.jar
snappy-java-1.1.2.6.jar
zkclient-0.10.jar
zookeeper-3.4.10.jar

10.2.10.2.9.13 Kafka 0.10.2.0

connect-api-0.10.2.0.jar
connect-json-0.10.2.0.jar
jackson-annotations-2.8.0.jar
jackson-core-2.8.5.jar
jackson-databind-2.8.5.jar
jopt-simple-5.0.3.jar
kafka_2.11-0.10.2.0.jar
kafka-clients-0.10.2.0.jar
log4j-1.2.17.jar
lz4-1.3.0.jar
metrics-core-2.2.0.jar
scala-library-2.11.8.jar
scala-parser-combinators_2.11-1.0.4.jar
slf4j-api-1.7.21.jar
slf4j-log4j12-1.7.21.jar
snappy-java-1.1.2.6.jar
zkclient-0.10.jar
zookeeper-3.4.9.jar

10.2.10.2.9.14 Kafka 0.10.2.0

connect-api-0.10.1.1.jar
connect-json-0.10.1.1.jar
jackson-annotations-2.6.0.jar
jackson-core-2.6.3.jar
jackson-databind-2.6.3.jar
jline-0.9.94.jar
jopt-simple-4.9.jar
kafka_2.11-0.10.1.1.jar
kafka-clients-0.10.1.1.jar
log4j-1.2.17.jar
lz4-1.3.0.jar
metrics-core-2.2.0.jar
netty-3.7.0.Final.jar
scala-library-2.11.8.jar
scala-parser-combinators_2.11-1.0.4.jar
slf4j-api-1.7.21.jar
slf4j-log4j12-1.7.21.jar
snappy-java-1.1.2.6.jar
zkclient-0.9.jar
zookeeper-3.4.8.jar

10.2.10.2.9.15 Kafka 0.10.0.0

activation-1.1.jar
connect-api-0.10.0.0.jar
connect-json-0.10.0.0.jar
jackson-annotations-2.6.0.jar
jackson-core-2.6.3.jar
jackson-databind-2.6.3.jar
jline-0.9.94.jar
jopt-simple-4.9.jar
junit-3.8.1.jar
kafka_2.11-0.10.0.0.jar
kafka-clients-0.10.0.0.jar
log4j-1.2.15.jar
lz4-1.3.0.jar
mail-1.4.jar
metrics-core-2.2.0.jar
netty-3.7.0.Final.jar
scala-library-2.11.8.jar
scala-parser-combinators_2.11-1.0.4.jar
slf4j-api-1.7.21.jar
slf4j-log4j12-1.7.21.jar
snappy-java-1.1.2.4.jar
zkclient-0.8.jar
zookeeper-3.4.6.jar

10.2.10.2.9.16 Kafka 0.9.0.1

activation-1.1.jar
connect-api-0.9.0.1.jar
connect-json-0.9.0.1.jar
jackson-annotations-2.5.0.jar
jackson-core-2.5.4.jar
jackson-databind-2.5.4.jar
jline-0.9.94.jar
jopt-simple-3.2.jar
junit-3.8.1.jar
kafka_2.11-0.9.0.1.jar

kafka-clients-0.9.0.1.jar
log4j-1.2.15.jar
lz4-1.2.0.jar
mail-1.4.jar
metrics-core-2.2.0.jar
netty-3.7.0.Final.jar
scala-library-2.11.7.jar
scala-parser-combinators_2.11-1.0.4.jar
scala-xml_2.11-1.0.4.jar
slf4j-api-1.7.6.jar
slf4j-log4j12-1.7.6.jar
snappy-java-1.1.1.7.jar
zkclient-0.7.jar
zookeeper-3.4.6.jar

- [Confluent Dependencies](#)

10.2.10.2.9.16.1 Confluent Dependencies

Note

The Confluent dependencies listed below are for the Kafka Connect Avro Converter and the associated Avro Schema Registry client. When integrated with Confluent Kafka Connect, the below dependencies are required in addition to the Kafka Connect dependencies for the corresponding Kafka version which are listed in the previous sections.

- [Confluent 6.2.0](#)
- [Confluent 6.1.0](#)
- [Confluent 6.0.0](#)
- [Confluent 5.5.0](#)
- [Confluent 5.4.0](#)
- [Confluent 5.3.0](#)
- [Confluent 5.2.1](#)
- [Confluent 5.1.3](#)
- [Confluent 5.0.3](#)
- [Confluent 4.1.2](#)

10.2.10.2.9.16.1.1 Confluent 6.2.0

avro-1.10.1.jar
commons-compress-1.20.jar
common-utils-6.2.0.jar
connect-api-6.2.0-ccs.jar
connect-json-6.2.0-ccs.jar
jackson-annotations-2.10.5.jar
jackson-core-2.11.3.jar
jackson-databind-2.10.5.1.jar
jackson-datatype-jdk8-2.10.5.jar
jakarta.annotation-api-1.3.5.jar
jakarta.inject-2.6.1.jar
jakarta.ws.rs-api-2.1.6.jar
javax.ws.rs-api-2.1.1.jar

jersey-common-2.34.jar
kafka-avro-serializer-6.2.0.jar
kafka-clients-6.2.0-ccs.jar
kafka-connect-avro-converter-6.2.0.jar
kafka-connect-avro-data-6.2.0.jar
kafka-schema-registry-client-6.2.0.jar
kafka-schema-serializer-6.2.0.jar
lz4-java-1.7.1.jar
osgi-resource-locator-1.0.3.jar
slf4j-api-1.7.30.jar
snappy-java-1.1.8.1.jar
swagger-annotations-1.6.2.jar
zstd-jni-1.4.9-1.jar

10.2.10.2.9.16.1.2 Confluent 6.1.0

avro-1.9.2.jar
commons-compress-1.19.jar
common-utils-6.1.0.jar
connect-api-6.1.0-ccs.jar
connect-json-6.1.0-ccs.jar
jackson-annotations-2.10.5.jar
jackson-core-2.10.2.jar
jackson-databind-2.10.5.1.jar
jackson-datatype-jdk8-2.10.5.jar
jakarta.annotation-api-1.3.5.jar
jakarta.inject-2.6.1.jar
jakarta.ws.rs-api-2.1.6.jar
javax.ws.rs-api-2.1.1.jar
jersey-common-2.31.jar
kafka-avro-serializer-6.1.0.jar
kafka-clients-6.1.0-ccs.jar
kafka-connect-avro-converter-6.1.0.jar
kafka-connect-avro-data-6.1.0.jar
kafka-schema-registry-client-6.1.0.jar
kafka-schema-serializer-6.1.0.jar
lz4-java-1.7.1.jar
osgi-resource-locator-1.0.3.jar
slf4j-api-1.7.30.jar
snappy-java-1.1.7.7.jar
swagger-annotations-1.6.2.jar
zstd-jni-1.4.5-6.jar

10.2.10.2.9.16.1.3 Confluent 6.0.0

avro-1.9.2.jar
commons-compress-1.19.jar
common-utils-6.0.0.jar
connect-api-6.0.0-ccs.jar
connect-json-6.0.0-ccs.jar
jackson-annotations-2.10.5.jar
jackson-core-2.10.2.jar
jackson-databind-2.10.5.jar
jackson-datatype-jdk8-2.10.5.jar
jakarta.annotation-api-1.3.5.jar
jakarta.inject-2.6.1.jar
jakarta.ws.rs-api-2.1.6.jar
javax.ws.rs-api-2.1.1.jar
jersey-common-2.30.jar
kafka-avro-serializer-6.0.0.jar
kafka-clients-6.0.0-ccs.jar
kafka-connect-avro-converter-6.0.0.jar
kafka-connect-avro-data-6.0.0.jar

kafka-schema-registry-client-6.0.0.jar
kafka-schema-serializer-6.0.0.jar
lz4-java-1.7.1.jar
osgi-resource-locator-1.0.3.jar
slf4j-api-1.7.30.jar
snappy-java-1.1.7.3.jar
swagger-annotations-1.6.2.jar
zstd-jni-1.4.4-7.jar

10.2.10.2.9.16.1.4 Confluent 5.5.0

avro-1.9.2.jar
classmate-1.3.4.jar
common-config-5.5.0.jar
commons-compress-1.19.jar
commons-lang3-3.2.1.jar
common-utils-5.5.0.jar
connect-api-5.5.0-ccs.jar
connect-json-5.5.0-ccs.jar
guava-18.0.jar
hibernate-validator-6.0.17.Final.jar
jackson-annotations-2.10.2.jar
jackson-core-2.10.2.jar
jackson-databind-2.10.2.jar
jackson-dataformat-yaml-2.4.5.jar
jackson-datatype-jdk8-2.10.2.jar
jackson-datatype-joda-2.4.5.jar
jakarta.annotation-api-1.3.5.jar
jakarta.el-3.0.2.jar
jakarta.el-api-3.0.3.jar
jakarta.inject-2.6.1.jar
jakarta.validation-api-2.0.2.jar
jakarta.ws.rs-api-2.1.6.jar
javax.ws.rs-api-2.1.1.jar
jboss-logging-3.3.2.Final.jar
jersey-bean-validation-2.30.jar
jersey-client-2.30.jar
jersey-common-2.30.jar
jersey-media-jaxb-2.30.jar
jersey-server-2.30.jar
joda-time-2.2.jar
kafka-avro-serializer-5.5.0.jar
kafka-clients-5.5.0-ccs.jar
kafka-connect-avro-converter-5.5.0.jar
kafka-connect-avro-data-5.5.0.jar
kafka-schema-registry-client-5.5.0.jar
kafka-schema-serializer-5.5.0.jar
lz4-java-1.7.1.jar
osgi-resource-locator-1.0.3.jar
slf4j-api-1.7.30.jar
snakeyaml-1.12.jar
snappy-java-1.1.7.3.jar
swagger-annotations-1.5.22.jar
swagger-core-1.5.3.jar
swagger-models-1.5.3.jar
zstd-jni-1.4.4-7.jar

10.2.10.2.9.16.1.5 Confluent 5.4.0

avro-1.9.1.jar
common-config-5.4.0.jar
commons-compress-1.19.jar
commons-lang3-3.2.1.jar

common-utils-5.4.0.jar
connect-api-5.4.0-ccs.jar
connect-json-5.4.0-ccs.jar
guava-18.0.jar
jackson-annotations-2.9.10.jar
jackson-core-2.9.9.jar
jackson-databind-2.9.10.1.jar
jackson-dataformat-yaml-2.4.5.jar
jackson-datatype-jdk8-2.9.10.jar
jackson-datatype-joda-2.4.5.jar
javax.ws.rs-api-2.1.1.jar
joda-time-2.2.jar
kafka-avro-serializer-5.4.0.jar
kafka-clients-5.4.0-ccs.jar
kafka-connect-avro-converter-5.4.0.jar
kafka-schema-registry-client-5.4.0.jar
lz4-java-1.6.0.jar
slf4j-api-1.7.28.jar
snakeyaml-1.12.jar
snappy-java-1.1.7.3.jar
swagger-annotations-1.5.22.jar
swagger-core-1.5.3.jar
swagger-models-1.5.3.jar
zstd-jni-1.4.3-1.jar

10.2.10.2.9.16.1.6 Confluent 5.3.0

audience-annotations-0.5.0.jar
avro-1.8.1.jar
common-config-5.3.0.jar
commons-compress-1.8.1.jar
common-utils-5.3.0.jar
connect-api-5.3.0-ccs.jar
connect-json-5.3.0-ccs.jar
jackson-annotations-2.9.0.jar
jackson-core-2.9.9.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.9.9.jar
jackson-datatype-jdk8-2.9.9.jar
jackson-mapper-asl-1.9.13.jar
javax.ws.rs-api-2.1.1.jar
jline-0.9.94.jar
jsr305-3.0.2.jar
kafka-avro-serializer-5.3.0.jar
kafka-clients-5.3.0-ccs.jar
kafka-connect-avro-converter-5.3.0.jar
kafka-schema-registry-client-5.3.0.jar
lz4-java-1.6.0.jar
netty-3.10.6.Final.jar
paranamer-2.7.jar
slf4j-api-1.7.26.jar
snappy-java-1.1.1.3.jar
spotbugs-annotations-3.1.9.jar
xz-1.5.jar
zkclient-0.10.jar
zookeeper-3.4.14.jar
zstd-jni-1.4.0-1.jar

10.2.10.2.9.16.1.7 Confluent 5.2.1

audience-annotations-0.5.0.jar
avro-1.8.1.jar
common-config-5.2.1.jar

commons-compress-1.8.1.jar
common-utils-5.2.1.jar
connect-api-2.2.0-cp2.jar
connect-json-2.2.0-cp2.jar
jackson-annotations-2.9.0.jar
jackson-core-2.9.8.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.9.8.jar
jackson-datatype-jdk8-2.9.8.jar
jackson-mapper-asl-1.9.13.jar
javax.ws.rs-api-2.1.1.jar
jline-0.9.94.jar
kafka-avro-serializer-5.2.1.jar
kafka-clients-2.2.0-cp2.jar
kafka-connect-avro-converter-5.2.1.jar
kafka-schema-registry-client-5.2.1.jar
lz4-java-1.5.0.jar
netty-3.10.6.Final.jar
paranamer-2.7.jar
slf4j-api-1.7.25.jar
snappy-java-1.1.1.3.jar
xz-1.5.jar
zkclient-0.10.jar
zookeeper-3.4.13.jar
zstd-jni-1.3.8-1.jar

10.2.10.2.9.16.1.8 Confluent 5.1.3

audience-annotations-0.5.0.jar
avro-1.8.1.jar
common-config-5.1.3.jar
commons-compress-1.8.1.jar
common-utils-5.1.3.jar
connect-api-2.1.1-cp3.jar
connect-json-2.1.1-cp3.jar
jackson-annotations-2.9.0.jar
jackson-core-2.9.8.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.9.8.jar
jackson-mapper-asl-1.9.13.jar
javax.ws.rs-api-2.1.1.jar
jline-0.9.94.jar
kafka-avro-serializer-5.1.3.jar
kafka-clients-2.1.1-cp3.jar
kafka-connect-avro-converter-5.1.3.jar
kafka-schema-registry-client-5.1.3.jar
lz4-java-1.5.0.jar
netty-3.10.6.Final.jar
paranamer-2.7.jar
slf4j-api-1.7.25.jar
snappy-java-1.1.1.3.jar
xz-1.5.jar
zkclient-0.10.jar
zookeeper-3.4.13.jar
zstd-jni-1.3.7-1.jar

10.2.10.2.9.16.1.9 Confluent 5.0.3

audience-annotations-0.5.0.jar
avro-1.8.1.jar
common-config-5.0.3.jar
commons-compress-1.8.1.jar
common-utils-5.0.3.jar

```
connect-api-2.0.1-cp4.jar
connect-json-2.0.1-cp4.jar
jackson-annotations-2.9.0.jar
jackson-core-2.9.7.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.9.7.jar
jackson-mapper-asl-1.9.13.jar
javax.ws.rs-api-2.1.jar
jline-0.9.94.jar
kafka-avro-serializer-5.0.3.jar
kafka-clients-2.0.1-cp4.jar
kafka-connect-avro-converter-5.0.3.jar
kafka-schema-registry-client-5.0.3.jar
lz4-java-1.4.1.jar
netty-3.10.6.Final.jar
paranamer-2.7.jar
slf4j-api-1.7.25.jar
snappy-java-1.1.1.3.jar
xz-1.5.jar
zkclient-0.10.jar
zookeeper-3.4.13.jar
```

10.2.10.2.9.16.1.10 Confluent 4.1.2

```
avro-1.8.1.jar
common-config-4.1.2.jar
commons-compress-1.8.1.jar
common-utils-4.1.2.jar
connect-api-1.1.1-cp1.jar
connect-json-1.1.1-cp1.jar
jackson-annotations-2.9.0.jar
jackson-core-2.9.6.jar
jackson-core-asl-1.9.13.jar
jackson-databind-2.9.6.jar
jackson-mapper-asl-1.9.13.jar
jline-0.9.94.jar
kafka-avro-serializer-4.1.2.jar
kafka-clients-1.1.1-cp1.jar
kafka-connect-avro-converter-4.1.2.jar
kafka-schema-registry-client-4.1.2.jar
log4j-1.2.16.jar
lz4-java-1.4.1.jar
netty-3.10.5.Final.jar
paranamer-2.7.jar
slf4j-api-1.7.25.jar
slf4j-log4j12-1.6.1.jar
snappy-java-1.1.1.3.jar
xz-1.5.jar
zkclient-0.10.jar
zookeeper-3.4.10.jar
```

10.2.10.3 Apache Kafka REST Proxy

The Kafka REST Proxy Handler to stream messages to the Kafka REST Proxy distributed by Confluent.

This chapter describes how to use the Kafka REST Proxy Handler.

- [Overview](#)
- [Setting Up and Starting the Kafka REST Proxy Handler Services](#)

- [Consuming the Records](#)
- [Performance Considerations](#)
- [Kafka REST Proxy Handler Metacolumns Template Property](#)

10.2.10.3.1 Overview

The Kafka REST Proxy Handler allows Kafka messages to be streamed using an HTTPS protocol. The use case for this functionality is to stream Kafka messages from an Oracle GoldenGate On Premises installation to cloud or alternately from cloud to cloud.

The Kafka REST proxy provides a RESTful interface to a Kafka cluster. It makes it easy for you to:

- produce and consume messages,
- view the state of the cluster,
- and perform administrative actions without using the native Kafka protocol or clients.

Kafka REST Proxy is part of the Confluent Open Source and Confluent Enterprise distributions. It is not available in the Apache Kafka distribution. To access Kafka through the REST proxy, you have to install the Confluent Kafka version see <https://docs.confluent.io/current/kafka-rest/docs/index.html>.

10.2.10.3.2 Setting Up and Starting the Kafka REST Proxy Handler Services

You have several installation formats to choose from including ZIP or tar archives, Docker, and Packages.

- [Using the Kafka REST Proxy Handler](#)
- [Downloading the Dependencies](#)
- [Classpath Configuration](#)
- [Kafka REST Proxy Handler Configuration](#)
- [Review a Sample Configuration](#)
- [Security](#)
- [Generating a Keystore or Truststore](#)
- [Using Templates to Resolve the Topic Name and Message Key](#)
- [Kafka REST Proxy Handler Formatter Properties](#)

10.2.10.3.2.1 Using the Kafka REST Proxy Handler

You must download and install the Confluent Open Source or Confluent Enterprise Distribution because the Kafka REST Proxy is not included in Apache, Cloudera, or Hortonworks. You have several installation formats to choose from including ZIP or TAR archives, Docker, and Packages.

The Kafka REST Proxy has dependency on ZooKeeper, Kafka, and the Schema Registry

10.2.10.3.2.2 Downloading the Dependencies

You can review and download the Jersey RESTful Web Services in Java client dependency from:

<https://eclipse-ee4j.github.io/jersey/>.

You can review and download the Jersey Apache Connector dependencies from the maven repository: <https://mvnrepository.com/artifact/org.glassfish.jersey.connectors/jersey-apache-connector>

10.2.10.3.2.3 Classpath Configuration

The Kafka REST Proxy handler uses the Jersey project jersey-client version 2.27 and jersey-connectors-apache version 2.27 to connect to Kafka. Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) does not include the required dependencies so you must obtain them, see [Downloading the Dependencies](#).

You have to configure these dependencies using the `gg.classpath` property in the Java Adapter properties file. This is an example of a correctly configured classpath for the Kafka REST Proxy Handler:

```
gg.classpath=dirprm:
{path_to_jersey_client_jars}/jaxrs-ri/lib/*:{path_to_jersey_client_jars}
/jaxrs-ri/api/*
:{path_to_jersey_client_jars}/jaxrs-ri/ext/*:{path_to_jersey_client_jars}
/connector/*
```

10.2.10.3.2.4 Kafka REST Proxy Handler Configuration

The following are the configurable values for the Kafka REST Proxy Handler. Oracle recommend that you store the Kafka REST Proxy properties file in the Oracle GoldenGate dirprm directory.

To enable the selection of the Kafka REST Proxy Handler, you must first configure the handler type by specifying `gg.handler.name.type=kafkarestproxy` and the other Kafka REST Proxy Handler properties as follows:

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.type</code>	Required	kafkarestproxy	None	The configuration to select the Kafka REST Proxy Handler.
<code>gg.handler.name.topicMappingTemplate</code>	Required	A template string value to resolve the Kafka topic name at runtime.	None	See Using Templates to Resolve the Topic Name and Message Key .
<code>gg.handler.name.keyMappingTemplate</code>	Required	A template string value to resolve the Kafka message key at runtime.	None	See Using Templates to Resolve the Topic Name and Message Key .
<code>gg.handler.name.postDataUrl</code>	Required	The Listener address of the Rest Proxy.	None	Set to the URL of the Kafka REST proxy.
<code>gg.handler.name.format</code>	Required	avro json	None	Set to the REST proxy payload data format

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.payloadsize</code>	Optional	A value representing the payload size in mega bytes.	5MB	Set to the maximum size of the payload of the HTTP messages.
<code>gg.handler.name.apiVersion</code>	Optional	v1 v2	v2	Sets the API version to use.
<code>gg.handler.name.mode</code>	Optional	op tx	op	Sets how operations are processed. In op mode, operations are processed as they are received. In tx mode, operations are cached and processed at the transaction commit.
<code>gg.handler.name.trustStore</code>	Optional	Path to the truststore.	None	Path to the truststore file that holds certificates from trusted certificate authorities (CA). These CAs are used to verify certificates presented by the server during an SSL connection, see Generating a Keystore or Truststore .
<code>gg.handler.name.trustStorePassword</code>	Optional	Password of the truststore.	None	The truststore password.
<code>gg.handler.name.keyStore</code>	Optional	Path to the keystore.	None	Path to the keystore file that the private key and identity certificate, which are presented to other parties (server or client) to verify its identity, see Generating a Keystore or Truststore .
<code>gg.handler.name.keyStorePassword</code>	Optional	Password of the keystore.	None	The keystore password.
<code>gg.handler.name.proxy</code>	Optional	http:// host:port	None	Proxy URL in the following format: http:// host:port

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.proxyUserName</code>	Optional	Any string.	None	The proxy user name.
<code>gg.handler.name.proxyPassword</code>	Optional	Any string.	None	The proxy password.
<code>gg.handler.name.readTimeout</code>	Optional	Integer value.	None	The amount of time allowed for the server to respond.
<code>gg.handler.name.connectionTimeout</code>	Optional	Integer value.	None	The amount of time to wait to establish the connection to the host.

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler.name .format.metaColumnsTemplate	Optional	\${alltokens} \${token} \$ {env} \${sys} \${javaprop} \${optype} \$ {position} \$ {timestamp} \$ {catalog} \$ {schema} \$ {table} \$ {objectname} \${csn} \$ {xid} \$ {currenttimestamp} mp} \$ {opseqno} \$ {timestampmicro} } \$ {currenttimestamp} mpmicro} \${txind} \$ {primarykeycolumns} mns} \$ {currenttimestamp} mpiso8601}\$ {static}\$ {segno} \$ {rba}	None	\${alltokens} \${token} \$ {env} \${sys} \${javaprop} \${optype} \$ {position} \$ {timestamp} \$ {catalog} \$ {schema} \$ {table} \$ {objectname} \${csn} \$ {xid} \$ {currenttimestamp} mp} \$ {opseqno} \$ {timestampmicro} } \$ {currenttimestamp} mpmicro} \${txind} \$ {primarykeycolumns} mns} \$ {currenttimestamp} mpiso8601}\$ {static}\$ {segno} \$ {rba}
				It is a comma-delimited string consisting of one or more templated values that represent the template. For more information about the Metacolumn keywords, see Metacolumn Keywords . This is an example that would produce a list of metacolumns: \${optype}, \$ {token.ROWID}, \$ {sys.username}, \$ {currenttimestamp}

See [Using Templates to Resolve the Stream Name and Partition Name](#) for more information.

10.2.10.3.2.5 Review a Sample Configuration

The following is a sample configuration for the Kafka REST Proxy Handler from the Java Adapter properties file:

```
gg.handlerlist=kafkarestproxy

#The handler properties
gg.handler.kafkarestproxy.type=kafkarestproxy
#The following selects the topic name based on the fully qualified table name
gg.handler.kafkarestproxy.topicMappingTemplate=${fullyQualifiedTableName}
#The following selects the message key using the concatenated primary keys
gg.handler.kafkarestproxy.keyMappingTemplate=${primaryKeys}
gg.handler.kafkarestproxy.postDataUrl=http://localhost:8083
gg.handler.kafkarestproxy.apiVersion=v1
gg.handler.kafkarestproxy.format=json
gg.handler.kafkarestproxy.payloadsize=1
gg.handler.kafkarestproxy.mode=tx

#Server auth properties
#gg.handler.kafkarestproxy.trustStore=/keys/truststore.jks
#gg.handler.kafkarestproxy.trustStorePassword=test1234
#Client auth properties
#gg.handler.kafkarestproxy.keyStore=/keys/keystore.jks
#gg.handler.kafkarestproxy.keyStorePassword=test1234

#Proxy properties
#gg.handler.kafkarestproxy.proxy=http://proxyurl:80
#gg.handler.kafkarestproxy.proxyUserName=username
#gg.handler.kafkarestproxy.proxyPassword=password

#The MetaColumnTemplate formatter properties
gg.handler.kafkarestproxy.format.metaColumnsTemplate=${optype},${timestampmicro},${currenttimestampmicro}
```

10.2.10.3.2.6 Security

Security is possible between the following:

- Kafka REST Proxy clients and the Kafka REST Proxy server. The Oracle GoldenGate REST Proxy Handler is a Kafka REST Proxy client.
- The Kafka REST Proxy server and Kafka Brokers. Oracle recommends that you thoroughly review the security documentation and configuration of the Kafka REST Proxy server, see <https://docs.confluent.io/current/kafka-rest/docs/index.html>

REST Proxy supports SSL for securing communication between clients and the Kafka REST Proxy Handler. To configure SSL:

1. Generate a keystore using the scripts, see [Generating a Keystore or Truststore](#).
2. Update the Kafka REST Proxy server configuration in the `kafka-rest.properties` file with these properties:

```
listeners=https://hostname:8083
confluent.rest.auth.propagate.method=SSL
```

```
Configuration Options for HTTPS
ssl.client.auth=true
ssl.keystore.location={keystore_file_path}/server.keystore.jks
ssl.keystore.password=test1234
ssl.key.password=test1234
```

```

ssl.truststore.location={keystore_file_path}/server.truststore.jks
ssl.truststore.password=test1234
ssl.keystore.type=JKS
ssl.truststore.type=JKS
ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1

```

3. Restart your server.

To disable mutual authentication, you update the `ssl.client.auth=` property from `true` to `false`.

10.2.10.3.2.7 Generating a Keystore or Truststore

Generating a Truststore

You execute this script to generate the `ca-cert`, `ca-key`, and `truststore.jks` truststore files.

```

#!/bin/bash
PASSWORD=password
CLIENT_PASSWORD=password
VALIDITY=365

```

Then you generate a CA as in this example:

```

openssl req -new -x509 -keyout ca-key -out ca-cert -days $VALIDITY -passin pass:$PASSWORD
-passout pass:$PASSWORD -subj "/C=US/ST=CA/L=San Jose/O=Company/OU=Org/CN=FQDN"
-nodes

```

Lastly, you add the CA to the server's truststore using `keytool`:

```

keytool -keystore truststore.jks -alias CARoot -import -file ca-cert -storepass $PASSWORD
-keypass $PASSWORD

```

Generating a Keystore

You run this script and pass the `fqdn` as argument to generate the `ca-cert.srl`, `cert-file`, `cert-signed`, and `keystore.jks` keystore files.

```

#!/bin/bash
PASSWORD=password
VALIDITY=365

if [ $# -lt 1 ];
then
echo "`basename $0` host fqdn|user_name|app_name"
exit 1
fi

CNAME=$1
ALIAS=`echo $CNAME|cut -f1 -d"."`

```

Then you generate the keystore with `keytool` as in this example:

```

keytool -noprompt -keystore keystore.jks -alias $ALIAS -keyalg RSA -validity $VALIDITY
-genkey -dname "CN=$CNAME,OU=BDP,O=Company,L=San Jose,S=CA,C=US" -
storepass $PASSWORD
-keypass $PASSWORD

```

Next, you sign all the certificates in the keystore with the CA:

```

keytool -keystore keystore.jks -alias $ALIAS -certreq -file cert-file -storepass
$PASSWORDopenssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-

```

```
signed -days $VALIDITY
-Ccreateserial -passin pass:$PASSWORD
```

Lastly, you import both the CA and the signed certificate into the keystore:

```
keytool -keystore keystore.jks -alias CARoot -import -file ca-cert -storepass
$PASSWORDkeytool -keystore keystore.jks -alias $ALIAS -import -file cert-signed -
storepass
$PASSWORD
```

10.2.10.3.2.8 Using Templates to Resolve the Topic Name and Message Key

The Kafka REST Proxy Handler provides functionality to resolve the topic name and the message key at runtime using a template configuration value. Templates allow you to configure static values and keywords. Keywords are used to dynamically replace the keyword with the context of the current processing. The templates use the following configuration properties:

```
gg.handler.name.topicMappingTemplate
gg.handler.name.keyMappingTemplate
```

Template Modes

The Kafka REST Proxy Handler can be configured to send one message per operation (insert, update, delete). Alternatively, it can be configured to group operations into messages at the transaction level.

For more information about the Template Keywords, see [Template Keywords](#).

Example Templates

The following describes example template configuration values and the resolved values.

Example Template	Resolved Value
<code>\${groupName}_\${fullyQualifiedTableName}</code>	KAFKA001_dbo.table1
<code>prefix_\${schemaName}_\${tableName}_suffix</code>	prefix_dbo_table1_suffix
<code>\${currentDate[yyyy-mm-dd hh:MM:ss.SSS]}</code>	2017-05-17 11:45:34.254

10.2.10.3.2.9 Kafka REST Proxy Handler Formatter Properties

The following are the configurable values for the Kafka REST Proxy Handler Formatter.

Table 10-12 Kafka REST Proxy Handler Formatter Properties

Properties	Optional/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.format.include</code> <code>OpType</code>	Optional	true false	true	Set to true to create a field in the output messages called <code>op_ts</code> . The value is an indicator of the type of source database operation (for example, I for insert, U for update, D for delete). Set to false to omit this field in the output.
<code>gg.handler.name</code> <code>.format.include</code> <code>OpTimestamp</code>	Optional	true false	true	Set to true to create a field in the output messages called <code>op_type</code> . The value is the operation timestamp (commit timestamp) from the source trail file. Set to false to omit this field in the output.
<code>gg.handler.name</code> <code>.format.include</code> <code>CurrentTimestamp</code>	Optional	true false	true	Set to true to create a field in the output messages called <code>current_ts</code> . The value is the current timestamp of when the handler processes the operation. Set to false to omit this field in the output.
<code>gg.handler.name</code> <code>.format.include</code> <code>Position</code>	Optional	true false	true	Set to true to create a field in the output messages called <code>pos</code> . The value is the position (sequence number + offset) of the operation from the source trail file. Set to false to omit this field in the output.

Table 10-12 (Cont.) Kafka REST Proxy Handler Formatter Properties

Properties	Optional/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.format.include</code> PrimaryKeys	Optional	true false	true	Set to true to create a field in the output messages called <code>primary_keys</code> . The value is an array of the column names of the primary key columns. Set to false to omit this field in the output.
<code>gg.handler.name</code> <code>.format.include</code> Tokens	Optional	true false	true	Set to true to include a map field in output messages. The key is <code>tokens</code> and the value is a map where the keys and values are the token keys and values from the Oracle GoldenGate source trail file. Set to false to suppress this field.
<code>gg.handler.name</code> <code>.format.insert0</code> pKey	Optional	Any string.	I	The value of the field <code>op_type</code> that indicates an insert operation.
<code>gg.handler.name</code> <code>.format.update0</code> pKey	Optional	Any string.	U	The value of the field <code>op_type</code> that indicates an update operation.
<code>gg.handler.name</code> <code>.format.delete0</code> pKey	Optional	Any string.	D	The value of the field <code>op_type</code> that indicates a delete operation.
<code>gg.handler.name</code> <code>.format.truncat</code> eOpKey	Optional	Any string.	T	The value of the field <code>op_type</code> that indicates a truncate operation.

Table 10-12 (Cont.) Kafka REST Proxy Handler Formatter Properties

Properties	Optional/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.format.treatAllColumnsAsStrings</code>	Optional	true false	false	Set to true treat all output fields as strings. Set to false and the handler maps the corresponding field type from the source trail file to the best corresponding Kafka data type.
<code>gg.handler.name</code> <code>.format.mapLargeNumbersAsStrings</code>	Optional	true false	false	Set to true and these fields are mapped as strings to preserve precision. This property is specific to the Avro Formatter; it cannot be used with other formatters.
<code>gg.handler.name</code> <code>.format.iso8601Format</code>	Optional	true false	false	Set to true to output the current date in the ISO8601 format.
<code>gg.handler.name</code> <code>.format.pkUpdateHandling</code>	Optional	abend update delete-insert	abend	It is only applicable if you are modeling row messages with the <code>. (gg.handler.name.format.messageFormatting=row</code> property. It is not applicable if you are modeling operations messages as the before and after images are propagated to the message with an update.

10.2.10.3.3 Consuming the Records

A simple way to consume data from Kafka topics using the Kafka REST Proxy Handler is Curl.

Consume JSON Data

1. Create a consumer for JSON data.

```
curl -k -X POST -H "Content-Type: application/vnd.kafka.v2+json"
https://localhost:8082/consumers/my_json_consumer
```


2. Subscribe to a topic.

```
curl -k -X POST -H "Content-Type: application/vnd.kafka.v2+json" --data
'{"topics":["topicname"]}' \
```

```
https://localhost:8082/consumers/my_json_consumer/instances/my_consumer_instance/
subscription
```

3. Consume records.

```
curl -k -X GET -H "Accept: application/vnd.kafka.json.v2+json" \
```

```
https://localhost:8082/consumers/my_json_consumer/instances/my_consumer_instance/
records
```

Consume Avro Data**1. Create a consumer for Avro data.**

```
curl -k -X POST -H "Content-Type: application/vnd.kafka.v2+json" \
--data '{"name": "my_consumer_instance", "format": "avro", "auto.offset.reset":
"earliest"}' \
```

```
https://localhost:8082/consumers/my_avro_consumer
```

2. Subscribe to a topic.

```
curl -k -X POST -H "Content-Type: application/vnd.kafka.v2+json" --data
'{"topics":["topicname"]}' \
```

```
https://localhost:8082/consumers/my_avro_consumer/instances/my_consumer_instance/
subscription
```

3. Consume records.

```
curl -X GET -H "Accept: application/vnd.kafka.avro.v2+json" \
```

```
https://localhost:8082/consumers/my_avro_consumer/instances/my_consumer_instance/
records
```

Note

If you are using `curl` from the machine hosting the REST proxy, then unset the `http_proxy` environmental variable before consuming the messages. If you are using `curl` from the local machine to get messages from the Kafka REST Proxy, then setting the `http_proxy` environmental variable may be required.

10.2.10.3.4 Performance Considerations

There are several configuration settings both for the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) configuration and in the Kafka producer that affects performance.

The Oracle GoldenGate parameter that has the greatest affect on performance is the Replicat `GROUPTRANSOPS` parameter. It allows Replicat to group multiple source transactions into a single target transaction. At transaction commit, the Kafka REST Proxy Handler POST's the data to the Kafka Producer.

Setting the Replicat `GROUPTRANSOPS` to a larger number allows the Replicat to call the POST less frequently improving performance. The default value for `GROUPTRANSOPS` is 1000 and performance can be improved by increasing the value to 2500, 5000, or even 10000.

10.2.10.3.5 Kafka REST Proxy Handler Metacolumns Template Property

Problems Starting Kafka REST Proxy server

The script to start the Kafka REST Proxy server appends its CLASSPATH to the environment CLASSPATH variable. If set, the environment CLASSPATH can contain JAR files that conflict with the correct execution of the Kafka REST Proxy server and may prevent it from starting. Oracle recommends that you unset the CLASSPATH environmental variable before started your Kafka REST Proxy server. Reset the CLASSPATH to "" to overcome the problem.

10.2.11 Apache Hive

Integrating with Hive

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) release does not include a Hive storage handler because the HDFS Handler provides all of the necessary Hive functionality.

You can create a Hive integration to create tables and update table definitions in case of DDL events. This is limited to data formatted in Avro Object Container File format. For more information, see [Writing in HDFS in Avro Object Container File Format](#) and [HDFS Handler Configuration](#).

For Hive to consume sequence files, the DDL creates Hive tables including STORED as sequencefile. The following is a sample create table script:

```
CREATE EXTERNAL TABLE table_name (  
    col1 string,  
    ...  
    ...  
    col2 string)  
ROW FORMAT DELIMITED  
STORED as sequencefile  
LOCATION '/path/to/hdfs/file';
```

Note

If files are intended to be consumed by Hive, then the `gg.handler.name.partitionByTable` property should be set to true.

Apache Hive Table Creation for Avro OCF

Use the following Apache Hive DDL to register external tables over the HDFS location that contains Avro OCF data:

```
CREATE EXTERNAL TABLE my_avro_ocf_table  
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'  
STORED AS  
INPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'  
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'  
LOCATION 'hdfs:///user/hive/schema/'  
TBLPROPERTIES (
```

```
'avro.schema.url' = 'hdfs:///user/hive/schema/QASOURCE.TCUSTMER.avsc'  
);
```

Note

- **LOCATION:** Points to the HDFS directory where Oracle GoldenGate writes .avro files.
- **avro.schema.url:** Path to the Avro (.avsc) schema file(s) that describes the records structure written by Oracle GoldenGate.

10.2.12 Apache Iceberg

Apache Iceberg is a high-performance table format for extremely large analytic tables. Apache Iceberg brings the reliability and simplicity of SQL tables to GG for DAA, while making it possible for engines, such as Spark, Trino, Flink, Presto, Hive, and Impala to safely work with the same tables, at the same time.

- [Detailed Functionality](#)
- [Configuration](#)
- [Configuration Templates](#)
- [Limitations](#)
- [Instantiating Oracle GoldenGate with an Initial Load](#)
- [Iceberg Version 3 Support](#)
- [Troubleshooting and Diagnostics](#)

10.2.12.1 Detailed Functionality

The Oracle GoldenGate Iceberg Replicat can replicate GoldenGate trail records to Iceberg tables.

The Iceberg open-table-format files could be written to local files, AWS Simple Storage Service(S3), Google Cloud Storage(GCS), or Azure DataLake Storage(ADLS).

- [Replication without a SQL Engine](#)
- [Iceberg File Format](#)
- [Iceberg Catalog](#)
- [Iceberg Specification](#)
- [Delete Files and Merge-On-Read \(MoR\)](#)
- [Operation Support](#)
- [Compressed Update Handling](#)
- [INSERTALLRECORDS Support](#)
- [Operation Aggregation](#)
- [Automatic Table Creation](#)
- [Iceberg Metadata Provider](#)
- [Iceberg Identifier Fields](#)

- [Primary Key Updates and Truncates](#)

10.2.12.1.1 Replication without a SQL Engine

Oracle GoldenGate Iceberg Replicat process does not require a SQL engine to replicate data to Iceberg tables.

It uses the Iceberg Java SDK along with object storage specific Java SDK to write data to Iceberg tables.

10.2.12.1.2 Iceberg File Format

The default file format for Iceberg data files and delete files is Parquet.

Oracle GoldenGate can be configured to write files in any of the following Iceberg supported file formats:

- Parquet (default)
- Avro
- ORC

10.2.12.1.3 Iceberg Catalog

Oracle GoldenGate supports the following Iceberg catalogs:

- Hadoop Catalog
- Nessie Catalog
- AWS Glue Catalog
- Polaris Catalog
- REST Catalog

10.2.12.1.4 Iceberg Specification

Oracle GoldenGate generates data files and delete files as per the Iceberg specification version 3 or version 2.

See <https://iceberg.apache.org/spec/>

10.2.12.1.5 Delete Files and Merge-On-Read (MoR)

Oracle GoldenGate generates Iceberg delete files for the UPDATE and DELETE operations.

Therefore, the Iceberg table property `write.update.mode` is always set to `merge-on-read`.

SQL engines should support merge-on-read to query tables replicated by Oracle GoldenGate.

Iceberg supports two types of delete files:

- **Equality Deletes:** The deleted records are identified by the equality of the values in the columns specified in the delete file.
- **Position Deletes:** The deleted records are identified by the position of the records in the Iceberg data file.
In the current release, Oracle GoldenGate uses Iceberg Equality Deletes to delete records from the Iceberg table.

This allows records to be deleted without looking up the position of the rows in the Iceberg data file.

Note

Contact Oracle support for use cases that require Iceberg Position Deletes.

- Oracle GoldenGate generates Iceberg delete files for the UPDATE and DELETE operations.

10.2.12.1.6 Operation Support

The Iceberg event handler supports the following operations:

- INSERT: Generates Iceberg data files for the insert operations.
- UPDATE: Generates Iceberg data files and delete files for update operations.
- DELETE: Generates Iceberg delete files for delete operations.
- TRUNCATE: Generates an Iceberg delete file with a condition as always true to truncate the target table.
This operation creates an empty Iceberg snapshot with no data files.

10.2.12.1.7 Compressed Update Handling

A compressed update record in the Oracle GoldenGate trail file contains values for the key columns and the modified columns.

An uncompressed update record contains values for all the columns.

Oracle GoldenGate trails may contain compressed or uncompressed update records. The default extract configuration writes compressed updates to the trail files.

If there are missing column values in the update operations, then Replicat will ABEND.

This behavior can be overridden by setting the parameter `gg.eventhandler.iceberg.abendOnMissingColumns=false` in the Replicat properties file.

When the parameter is set to false, Replicat will handle compressed updates by querying the previous values of the missing columns from the Iceberg table.

- [Lookup Missing values in Sparse Updates](#)

10.2.12.1.7.1 Lookup Missing values in Sparse Updates

The lookup of the missing values is an expensive operation and may impact the performance of the Replicat process.

By default, Oracle GoldenGate writes records to Iceberg in micro batches every ten minutes.

Every micro-batch for a table can potentially contain millions of rows.

Micro batches will be processed for every target table in concurrent threads.

Therefore, it is critical that sufficient JVM heap memory is allocated to the Replicat process.

The lookup is performed only for such rows that contain at least one missing value in the update operation.

Oracle GoldenGate will automatically create target tables. During auto-creation of tables, Oracle GoldenGate Replicat will enable creation of Iceberg metrics (min/max values) for all the identifier (key) columns.

The metrics are stored in the Iceberg metadata files.

Iceberg metrics helps speed up the lookup of the missing values in the UPDATE operations.

10.2.12.1.8 INSERTALLRECORDS Support

Iceberg event handler supports INSERTALLRECORDS parameter. See: <https://docs.oracle.com/en/middleware/goldengate/core/21.3/reference/insertallrecords.html#GUID-A1019C40-97BE-437B-9D80-7C99A9A6DB8E>. Set the INSERTALLRECORDS parameter in the Replicat parameter file (.prm).

Setting this property directs the Replicat process to generate Iceberg data files to append operation data into the Iceberg target table.

10.2.12.1.9 Operation Aggregation

Operation aggregation is the process of aggregating (merging/compressing) multiple operations on the same row into a single output operation based on a threshold.

Operation records are aggregated in-memory.

You can tune the frequency of apply interval using `gg.handler.iceberg.fileRollInterval` property, the default value is set to 15m (fifteen minutes).

The Replicat process will generate Iceberg data files and delete files for the aggregated operations.

10.2.12.1.10 Automatic Table Creation

Oracle GoldenGate Replicat will automatically create target tables if the target table does not exist.

10.2.12.1.11 Iceberg Metadata Provider

A new metadata provider for Iceberg is implemented to retrieve the Iceberg target table metadata.

Iceberg Metadata provider is auto configured and enabled by the Replicat process.

10.2.12.1.12 Iceberg Identifier Fields

The identifier fields in the Iceberg table are used to uniquely identify the rows in the Iceberg table.

During the automatic table creation, Oracle GoldenGate maps the key columns to Iceberg identifier fields.

Note

Iceberg tables without identifier fields are not supported in the current release.

10.2.12.1.13 Primary Key Updates and Truncates

- Primary key updates with missing column values will trigger files to be flushed to the Iceberg table before the flush interval.
This can result in small data files and delete files for the primary key update operation.

For workloads or tables with frequent primary key updates, Oracle recommends to generate trail files with uncompressed update records.

Oracle also recommends to set `gg.validate.keyupdate=true` for trail generated from Oracle source.

There is a known issue with Oracle extract to generate primary key update operations even though the key columns are not modified.
- A truncate operation will trigger files to be flushed to the Iceberg table before the flush interval.

10.2.12.2 Configuration

The configuration of the Iceberg replication properties is stored in the Replicat properties file.

- [Automatic Configuration](#)
- [Configuration for Iceberg Nessie Catalog](#)
- [Configuration for Iceberg AWS Glue Catalog](#)
- [Configuration for Iceberg Polaris Catalog](#)
- [Configuration for Iceberg REST Catalog](#)
- [Configuration for Iceberg Hadoop Catalog](#)

10.2.12.2.1 Automatic Configuration

Iceberg replication involves configuring multiple components, such as the File Writer Handler, and the target Iceberg Event Handler.

The Automatic Configuration functionality helps you to autoconfigure these components so that the manual configuration is minimal.

The properties modified by autoconfiguration is also logged in the handler log file.

To enable autoconfiguration to replicate to the Iceberg target, set the parameter `gg.target=iceberg`.

- [File Writer Configuration](#)
- [Iceberg Event Handler Configuration](#)

10.2.12.2.1.1 File Writer Configuration

The File Writer Handler name is pre-set to the value `iceberg` and its properties are automatically set to the required values for Iceberg.

10.2.12.2.1.2 Iceberg Event Handler Configuration

The Iceberg Event Handler name is pre-set to the value `iceberg`.

This topic details the configuration properties available for the Iceberg Event handler, the required ones must be changed to match your Iceberg configuration.

- [Common Iceberg Properties](#)
- [Iceberg Common Dependencies](#)
- [AWS Java SDK dependencies for Writing to AWS S3 \(s3:// Scheme\)](#)
- [Hadoop AWS SDK Dependencies for Writing to AWS S3 \(s3a:// Scheme\)](#)
- [Hadoop Google Cloud Storage SDK Dependencies for Writing to Google Cloud Storage \(GCS\)](#)
- [Google Cloud Storage SDK Dependencies for Writing to Google Cloud Storage \(GCS\)](#)
- [Hadoop Azure SDK Dependencies for Writing to Azure Data Lake \(ADLS\)](#)

10.2.12.2.1.2.1 Common Iceberg Properties

Iceberg can be configured to work with multiple catalogs and object stores.

The following are the common properties:

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .iceberg.wareho useLocation	Optional	String value.	None	Directory path to the Iceberg warehouse location excluding the object storage scheme. Example: /path/to/warehouse. This is a required property when using the hadoop catalog. For other Iceberg catalogs, warehouse location has a catalog specific requirement.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .iceberg.fileRo llInterval	Optional	The default unit of measure is milliseconds. You can stipulate ms, s, m, h to signify milliseconds, seconds, minutes, or hours respectively. Examples of legal values include 10000, 10000ms, 10s, 10m, or 1.5h. Values of 0 or less indicate that file rolling on time is turned off.	15m	The parameter determines how often the data will be pushed into the Iceberg warehouse. Use with caution, the higher this value is the more data will need to be stored in the memory of the Replicat process.

Note
Use the parameter to determine the frequency with which data is pushed into the Iceberg warehouse. Use with caution, the higher this value is the more data will need to be stored in the memory of the Replicat process.

Properties	Required/Optional	Legal Values	Default	Explanation

t
s
d
e
f
a
u
l
t
v
a
l
u
e
(
1
5
m
)
w
i
l
l
i
n
c
r
e
a
s
e
t
h
e
a
m
o
u
n
t
o
f
d
a
t
a
s
t
o
r
e
d
i
n
t
h

Properties	Required/Optional	Legal Values	Default	Explanation

e
i
n
t
e
r
n
a
l
m
e
m
o
r
y
o
f
t
h
e
R
e
p
l
i
c
a
t
.
T
h
i
s
c
a
n
c
a
u
s
e
o
u
t
o
f
m
e
m
o
r
y
e
r
r
o

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .iceberg.fileSystemScheme	Optional	String value.	file://	Warehouse scheme to indicate the Iceberg object storage location. Valid values are: file://, gs://, s3://, s3a://, abfss://. For more information, see File System Scheme .

rs and support the Replicator Catalyst interface to run source-to-target memory.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.catalogType	Optional	String value.	hadoop	Iceberg catalog type. Valid values are: hadoop, nessie, rest, glue, polaris.
gg.eventhandler.iceberg.fileFormat	Optional	parquet, orc, or avro.	parquet	Iceberg table file format to be used in target tables. Supported file formats: Parquet, Avro, and ORC.
gg.eventhandler.iceberg.icebergVersion	Optional	2 or 3	2	Specifies the Iceberg version for generated data and delete files. For more information, see Iceberg Version 3 Support.
gg.eventhandler.iceberg.icebergTableProperties	Optional	String value.	None	Path to a table properties file to specify additional Iceberg table properties to set to the target tables.
gg.eventhandler.iceberg.abandonMissingColumns	Optional	true or false.	true	When set to true and the UPDATE operation contains a missing value, Replicat will ABEND. When set to false, Replicat will not ABEND if UPDATE operations have missing column values. The missing columns values will be read by querying the target tables. This lookup may impact the performance of the Replicat process.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.abendOnSchemaChanges	Optional	true or false	true	When set to true and schema changes are detected, the replicat process will ABEND. User can manually update the target schema and set the configuration to false to proceed. When set to false, a warning message is logged for schema changes.
gg.validate.key update	Optional	true or false	false	If set to true, Replicat will validate key update operations (optype 115) and correct to normal update if no key values have changed.

- [File System Scheme](#)

10.2.12.2.1.2.1.1 File System Scheme

The gg.eventhandler.iceberg.fileSystemScheme property is used to specify the object storage scheme.

The following are the supported object storage schemes:

- file://: Local file system
- gs://: Google Cloud Storage
- s3://: AWS S3
- s3a://: AWS S3
- abfss://: Azure Data Lake Storage

10.2.12.2.1.2.2 Iceberg Common Dependencies

The following are the common Iceberg dependencies:

```
<dependencies>
  <!-- Common Iceberg dependencies START -->
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>3.4.2</version>
  </dependency>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-mapreduce-client-core</artifactId>
    <version>3.4.0</version>
  </dependency>
  <dependency>
    <groupId>org.apache.iceberg</groupId>
    <artifactId>iceberg-arrow</artifactId>
```

```
<version>1.10.1</version>
</dependency>
<dependency>
  <groupId>org.apache.iceberg</groupId>
  <artifactId>iceberg-core</artifactId>
  <version>1.10.1</version>
</dependency>
<dependency>
  <groupId>org.apache.iceberg</groupId>
  <artifactId>iceberg-data</artifactId>
  <version>1.10.1</version>
</dependency>
<dependency>
  <groupId>org.apache.iceberg</groupId>
  <artifactId>iceberg-parquet</artifactId>
  <version>1.10.1</version>
</dependency>
<dependency>
  <groupId>org.apache.iceberg</groupId>
  <artifactId>iceberg-gcp</artifactId>
  <version>1.10.1</version>
</dependency>
<dependency>
  <groupId>org.apache.iceberg</groupId>
  <artifactId>iceberg-aws</artifactId>
  <version>1.10.1</version>
</dependency>
<dependency>
  <groupId>org.apache.iceberg</groupId>
  <artifactId>iceberg-orc</artifactId>
  <version>1.10.1</version>
</dependency>
<dependency>
  <dependency>
    <groupId>org.apache.iceberg</groupId>
    <artifactId>iceberg-nessie</artifactId>
    <version>1.10.1</version>
  </dependency>
  <dependency>
    <groupId>org.apache.parquet</groupId>
    <artifactId>parquet-avro</artifactId>
    <version>1.16.0</version>
  </dependency>
  <dependency>
    <groupId>org.apache.parquet</groupId>
    <artifactId>parquet-column</artifactId>
    <version>1.16.0</version>
  </dependency>
  <dependency>
    <groupId>org.apache.parquet</groupId>
    <artifactId>parquet-variant</artifactId>
    <version>1.16.0</version>
  </dependency>
  <dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.20.1</version>
  </dependency>
  <dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>kms</artifactId>
    <version>2.30.19</version>
  </dependency>
</dependency>
```

```

    </dependency>
    <!-- Common Iceberg dependencies END -->
</dependencies>

```

You can download the dependencies from maven central using the script `download_dependencies.sh` in the `DependencyDownloader` directory.

Follow these steps:

1. Change directory to `DependencyDownloader`.
2. Edit `config_proxy.sh` if proxy configuration is required.
3. Run the script:

```
./download_dependencies.sh xmls/iceberg-common.xml
```

This script will download the dependencies and store them in the `iceberg-common` directory. `gg.classpath` can be configured to include the dependencies from the `iceberg-common` directory as follows: `gg.classpath=/path/to/DependencyDownloader/dependencies/iceberg-common/*`

10.2.12.2.1.2.3 AWS Java SDK dependencies for Writing to AWS S3 (s3:// Scheme)

The following are the Iceberg dependencies to write to AWS S3 using the `s3://` scheme:

```

<dependencies>
  <!-- s3:// scheme dependencies START -->
  <dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>s3</artifactId>
    <version>2.28.6</version>
  </dependency>
  <dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>sts</artifactId>
    <version>2.28.6</version>
  </dependency>
  <dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>glue</artifactId>
    <version>2.28.6</version>
  </dependency>
  <dependency>
    <groupId>software.amazon.awssdk</groupId>
    <artifactId>url-connection-client</artifactId>
    <version>2.28.6</version>
  </dependency>
  <!-- s3:// scheme dependencies END -->
</dependencies>

```

The dependencies can be downloaded from maven central using the script `download_dependencies.sh` in the `DependencyDownloader` directory.

Follow these steps:

- Change directory to `DependencyDownloader`.
- Edit `config_proxy.sh` if proxy configuration is required.
- Run the script: `./download_dependencies.sh xmls/iceberg-aws-java-sdk.xml`

This script will download the dependencies and store them in the iceberg-aws-java-sdk directory.

gg.classpath: can be configured to include the dependencies as follows:

```
gg.classpath=/path/to/DependencyDownloader/dependencies/iceberg-aws-java-sdk/*:/path/to/DependencyDownloader/dependencies/iceberg-common/*
```

10.2.12.2.1.2.4 Hadoop AWS SDK Dependencies for Writing to AWS S3 (s3a:// Scheme)

The following are the Iceberg dependencies to write to AWS S3 using the s3a:// scheme:

```
<dependencies>
  <!-- s3a:// scheme dependencies START -->
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-aws</artifactId>
    <version>3.4.2</version>
  </dependency>
  <!-- s3a:// scheme dependencies END -->
</dependencies>
```

You can download the dependencies from maven central using the script download_dependencies.sh in the DependencyDownloader directory.

Follow these steps:

- Change directory to DependencyDownloader.
- Edit config_proxy.sh if proxy configuration is required.
- Run the script:

```
./download_dependencies.sh xmls/iceberg-hadoop-aws.xml
```

This script will download the dependencies and store them in the iceberg-hadoop-aws directory.

gg.classpath can be configured to include the dependencies as follows:

```
gg.classpath=/path/to/DependencyDownloader/dependencies/iceberg-hadoop-aws/*:/path/to/DependencyDownloader/dependencies/iceberg-common/*
```

10.2.12.2.1.2.5 Hadoop Google Cloud Storage SDK Dependencies for Writing to Google Cloud Storage (GCS)

The following are the Iceberg dependencies to write to GCS using the Hadoop GCS SDK:

```
<dependencies>
  <!-- gs:// scheme dependencies START -->
  <dependency>
    <groupId>com.google.cloud.bigdataoss</groupId>
    <artifactId>gcs-connector</artifactId>
    <version>hadoop3-2.2.31</version>
  </dependency>
  <!-- gs:// scheme dependencies END -->
</dependencies>
```

The dependencies can be downloaded from maven central using the script download_dependencies.sh in the DependencyDownloader directory.

Follow these steps:

- Change directory to DependencyDownloader.
- Edit config_proxy.sh if proxy configuration is required.
- Run the script: ./download_dependencies.sh xmls/iceberg-hadoop-gcs.xml

This script will download the dependencies and store them in the iceberg-hadoop-gcs directory.

gg.classpath can be configured to include the dependencies as follows:

```
gg.classpath=/path/to/DependencyDownloader/dependencies/iceberg-hadoop-gcs/*:/
path/to/DependencyDownloader/dependencies/iceberg-common/*
```

10.2.12.2.1.2.6 Google Cloud Storage SDK Dependencies for Writing to Google Cloud Storage (GCS)

The following are the Iceberg dependencies to write to GCS using the Google Cloud Storage Java SDK:

```
<dependencies>
  <dependency>
    <groupId>com.google.cloud</groupId>
    <artifactId>google-cloud-storage</artifactId>
    <version>2.37.0</version>
  </dependency>
</dependencies>
```

The dependencies can be downloaded from maven central using the script download_dependencies.sh in the DependencyDownloader directory.

Follow these steps:

- Change directory to DependencyDownloader.
- Edit config_proxy.sh if proxy configuration is required.
- Run the script:

```
./download_dependencies.sh xmls/iceberg-gcs-java-sdk.xml
```

This script will download the dependencies and store them in the iceberg-gcs-java-sdk directory.

gg.classpath can be configured to include the dependencies as follows:

```
gg.classpath=/path/to/DependencyDownloader/dependencies/iceberg-hadoop-gcs/*:/path/to/
DependencyDownloader/dependencies/iceberg-common/*
```

10.2.12.2.1.2.7 Hadoop Azure SDK Dependencies for Writing to Azure Data Lake (ADLS)

The following are the Iceberg dependencies to write to ADLS using the Hadoop Azure Java SDK:

```
<dependencies>
  <!-- abfss:// scheme dependencies START -->
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-azure</artifactId>
    <version>3.4.2</version>
  </dependency>
  <!-- abfss:// scheme dependencies END -->
</dependencies>
```

The dependencies can be downloaded from maven central using the script `download_dependencies.sh` in the `DependencyDownloader` directory.

Follow these steps:

- Change directory to `DependencyDownloader`.
- Edit `config_proxy.sh` if proxy configuration is required.
- Run the script:

```
./download_dependencies.sh xmls/iceberg-hadoop-azure.xml
```

This script will download the dependencies and store them in the `iceberg-hadoop-azure` directory.

`gg.classpath`: can be configured to include the dependencies as follows:

```
gg.classpath=/path/to/DependencyDownloader/dependencies/iceberg-hadoop-azure/*:/path/to/DependencyDownloader/dependencies/iceberg-common/*
```

10.2.12.2.2 Configuration for Iceberg Nessie Catalog

- [Configuration for Nessie Catalog and AWS S3 s3:// Scheme](#)
- [Configuration for Nessie Catalog and AWS S3 s3a:// Scheme](#)
- [Configuration for Nessie Catalog and GCS gs:// Scheme](#)
- [Configuration for Nessie Catalog and Azure Data Lake Storage abfss:// Scheme](#)

10.2.12.2.2.1 Configuration for Nessie Catalog and AWS S3 s3:// Scheme

The following are the configuration properties for the Nessie catalog and AWS S3 object store using `s3://` scheme:

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.iceberg.catalogType</code>	Optional	String value.	hadoop	nessie
<code>gg.eventhandler.iceberg.nessieBranch</code>	Optional	String value.	main	Nessie Catalog branch name where the Iceberg table metadata exists.
<code>gg.eventhandler.iceberg.nessieUri</code>	Required	String value.	None	Nessie Catalog endpoint URI. Example: <code>http://<nessie-server>.com:10001/api/v2</code>

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .iceberg.fileSystemScheme	Optional	String value.	file://	File system scheme to indicate AWS S3 object storage location: s3://.
gg.eventhandler .iceberg.awsS3Region	Required	String value.	None	AWS S3 bucket region. Example: us-east-2.
gg.eventhandler .iceberg.awsS3Bucket	Required	String value.	None	AWS S3 bucket name that houses the Iceberg Warehouse.
gg.eventhandler .iceberg.awsAccessKeyId	Optional	String value.	None	AWS access key id for authentication.
gg.eventhandler .iceberg.awsSecretKey	Optional	String value.	None	AWS secret access key for authentication.
gg.eventhandler .iceberg.awsSessionToken	Optional	String value.	None	AWS session token for authentication.
gg.eventhandler .iceberg.awsRoleArn	Optional	String value.	None	AWS role ARN for authentication.
gg.eventhandler .iceberg.awsS3Endpoint	Optional	String value.	None	AWS S3 endpoint.
gg.eventhandler .iceberg.proxyServer	Optional	String value.	None	Proxy server to connect to the AWS S3 object storage.
gg.eventhandler .iceberg.proxyPort	Optional	String value.	80	Proxy server port to connect to the AWS S3 object storage.
gg.eventhandler .iceberg.awsS3PathStyleAccess	Optional	true or false.	None	Enables path-style access for S3-compatible storage, for example, domain.com/bucket instead of virtual-hosted style like bucket.domain.com). This is typically required when using non-AWS S3-compatible storage, such as Dell ECS or MinIO.

- [Classpath And Dependencies](#)
- [Sample Configuration for Nessie Catalog and AWS S3 s3:// Scheme](#)

10.2.12.2.2.1.1 Classpath And Dependencies

The Java classpath (gg.classpath) should include the following dependencies:

- Iceberg common dependencies
- AWS SDK dependencies for writing to AWS S3 (s3:// scheme)

10.2.12.2.2.1.2 Sample Configuration for Nessie Catalog and AWS S3 s3:// Scheme

```
gg.target=iceberg
gg.eventhandler.iceberg.warehouseLocation=/path/to/iceberg/tables
gg.classpath=DependencyDownloader/dependencies/iceberg-aws-java-sdk/
*:DependencyDownloader/dependencies/iceberg-common/*
gg.eventhandler.iceberg.catalogType=nessie
gg.eventhandler.iceberg.nessieBranch=main
gg.eventhandler.iceberg.nessieUri=http://<nessie-server>:10001/api/v2
gg.eventhandler.iceberg.fileSystemScheme=s3://
gg.eventhandler.iceberg.awsS3Region=us-east-2
gg.eventhandler.iceberg.awsS3Bucket=<s3-bucket>
gg.eventhandler.iceberg.awsAccessKeyId=<access-key-id>
gg.eventhandler.iceberg.awsSecretKey=<secret-key>
gg.eventhandler.iceberg.proxyServer=<proxy-server>
gg.eventhandler.iceberg.proxyPort=<proxy-port>
```

10.2.12.2.2.2 Configuration for Nessie Catalog and AWS S3 s3a:// Scheme

The following are the configuration properties for the Nessie catalog and AWS S3 object store using s3a:// scheme:

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.catalogType	Optional	String value.	hadoop	nessie.
gg.eventhandler.iceberg.nessieBranch	Optional	String value.	main	Nessie Catalog branch name where the Iceberg table metadata exists.
gg.eventhandler.iceberg.nessieUri	Required	String value.	None	Nessie Catalog endpoint URI. Example: http://<nessie-server>.com:10001/api/v2.
gg.eventhandler.iceberg.fileSystemScheme	Optional	String value.	file://	File system scheme to indicate AWS S3 object storage location: s3a://.
gg.eventhandler.iceberg.awsS3Bucket	Required	String value.	None	AWS S3 bucket name that houses the Iceberg Warehouse.
gg.eventhandler.iceberg.awsAccessKeyId	Required	String value.	None	AWS access key id for authentication.
gg.eventhandler.iceberg.awsSecretKey	Required	String value.	None	AWS secret access key for authentication.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.awsSessionToken	Optional	String value.	None	AWS session token for authentication.
gg.eventhandler.iceberg.proxyServer	Optional	String value.	None	Proxy server to connect to the AWS S3 object storage.
gg.eventhandler.iceberg.proxyPort	Optional	String value.	80	Proxy server port to connect to the AWS S3 object storage.

- [Classpath and Dependencies](#)
- [Sample Configuration for Nessie Catalog and AWS S3 s3a:// scheme](#)

10.2.12.2.2.2.1 Classpath and Dependencies

The Java classpath (gg.classpath) should include the following dependencies:

- Iceberg common dependencies
- Hadoop AWS SDK dependencies for writing to AWS S3 (s3a:// scheme)

10.2.12.2.2.2.2 Sample Configuration for Nessie Catalog and AWS S3 s3a:// scheme

```
gg.target=iceberg
gg.eventhandler.iceberg.warehouseLocation=/path/to/iceberg/tables
gg.classpath=DependencyDownloader/dependencies/iceberg-hadoop-aws/*:DependencyDownloader/dependencies/iceberg-common/*
gg.eventhandler.iceberg.catalogType=nessie
gg.eventhandler.iceberg.nessieBranch=main
gg.eventhandler.iceberg.nessieUri=http://<nessie-server>:10001/api/v2
gg.eventhandler.iceberg.fileSystemScheme=s3a://
gg.eventhandler.iceberg.awsS3Region=us-east-2
gg.eventhandler.iceberg.awsS3Bucket=<s3-bucket>
gg.eventhandler.iceberg.awsAccessKeyId=<access-key-id>
gg.eventhandler.iceberg.awsSecretKey=<secret-key>
gg.eventhandler.iceberg.proxyServer=<proxy-server>
gg.eventhandler.iceberg.proxyPort=<proxy-port>
```

10.2.12.2.2.3 Configuration for Nessie Catalog and GCS gs:// Scheme

The following are the configuration properties for the Nessie catalog and GCS object store using gs:// scheme:

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.catalogType	Optional	String value.	hadoop	nessie.
gg.eventhandler.iceberg.nessieBranch	Optional	String value.	main	Nessie Catalog branch name where the Iceberg table metadata exists.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.nessieUri	Required	String value.	None	Nessie Catalog endpoint URI. Example: http://<nessie-server>.com:10001/api/v2.
gg.eventhandler.iceberg.fileSystemScheme	Optional	String value.	file://	File system scheme to indicate GCS object storage location: gs://.
gg.eventhandler.iceberg.gcpStorageBucket	Required	String value.	None	Google Cloud Storage bucket name that houses the Iceberg Warehouse.
gg.eventhandler.iceberg.gcpProjectId	Required	String value.	None	Sets the project-id of the Google Cloud project that houses the GCS bucket.
gg.eventhandler.iceberg.gcpServiceAccountJsonKeyFile	Required	String value.	None	Sets the path to the Google Service account key file.
gg.eventhandler.iceberg.proxyServer	Optional	String value.	None	Proxy server to connect to the GCS object storage.
gg.eventhandler.iceberg.proxyPort	Optional	String value.	80	Proxy server port to connect to the GCS object storage.

- [Classpath and Dependencies](#)
- [Sample Configuration for Nessie Catalog and GCS gs:// Scheme](#)

10.2.12.2.2.3.1 Classpath and Dependencies

The Java classpath (gg.classpath) should include the following dependencies:

- Iceberg common dependencies
- Hadoop Google Cloud Storage SDK dependencies for writing to Google Cloud Storage (GCS)

10.2.12.2.2.3.2 Sample Configuration for Nessie Catalog and GCS gs:// Scheme

```
gg.target=iceberg
gg.eventhandler.iceberg.warehouseLocation=/path/to/iceberg/tables
gg.classpath=DependencyDownloader/dependencies/iceberg-hadoop-gcs/*:DependencyDownloader/dependencies/iceberg-common/*
gg.eventhandler.iceberg.catalogType=nessie
gg.eventhandler.iceberg.nessieBranch=main
gg.eventhandler.iceberg.nessieUri=http://<nessie-server>:10001/api/v2
gg.eventhandler.iceberg.fileSystemScheme=gs://
gg.eventhandler.iceberg.gcpStorageBucket=<gcs-bucket>
gg.eventhandler.iceberg.gcpProjectId=<gcp-project-id>
gg.eventhandler.iceberg.gcpServiceAccountJsonKeyFile=<gcp-service-account-key-file>
gg.eventhandler.iceberg.proxyServer=<proxy-server>
gg.eventhandler.iceberg.proxyPort=<proxy-port>
```

10.2.12.2.2.4 Configuration for Nessie Catalog and Azure Data Lake Storage abfss:// Scheme

The following are the configuration properties for the Nessie catalog and Azure Data Lake Storage using abfss:// scheme:

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.catalogType	Optional	String value.	hadoop	nessie.
gg.eventhandler.iceberg.nessieBranch	Optional	String value.	main	Nessie Catalog branch name where the Iceberg table metadata exists.
gg.eventhandler.iceberg.nessieUri	Required	String value.	None	Nessie Catalog endpoint URI. Example: http://<nessie-server>.com:10001/api/v2.
gg.eventhandler.iceberg.fileSystemScheme	Optional	String value.	file://	File system scheme to indicate Azure Data Lake Storage location: abfss://.
gg.eventhandler.iceberg.azureAccountName	Required	String value.	None	Azure storage account name that contains the container for the Iceberg Warehouse.
gg.eventhandler.iceberg.azureContainer	Required	String value.	None	Azure storage account container name that houses the Iceberg Warehouse.
gg.eventhandler.iceberg.azureAccountKey	Required	String value.	None	Azure storage account key.
gg.eventhandler.iceberg.azureBlobEndpoint	Optional	String value.	<azureContainer>@<azureAccountName>.dfs.core.windows.net	Azure Storage service endpoint.
gg.eventhandler.iceberg.proxyServer	Optional	String value.	None	Proxy server to connect to the Azure object storage.
gg.eventhandler.iceberg.proxyPort	Optional	String value.	80	Proxy server port to connect to the Azure object storage.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.azureTenantId	Optional	String value.	None	Azure Tenant ID. Required only for EntraID auth (gg.eventhandler.iceberg.azureAccountKey has to be unset).
gg.eventhandler.iceberg.azureClientId	Optional	String value.	None	Azure EntraID Client ID. Required only for EntraID auth (gg.eventhandler.iceberg.azureAccountKey has to be unset)
gg.eventhandler.iceberg.azureClientSecret	Optional	String value.	None	Azure EntraID client secret. Required only for EntraID auth (gg.eventhandler.iceberg.azureAccountKey) has to be unset)

- [Classpath and Dependencies](#)
- [Sample Configuration for Nessie Catalog and ADLS abfss:// Scheme](#)
- [Nessie Namespace](#)

10.2.12.2.2.4.1 Classpath and Dependencies

The Java classpath (gg.classpath) should include the following dependencies:

- Iceberg common dependencies
- Hadoop Azure SDK dependencies for writing to Azure Data Lake (ADLS)

10.2.12.2.2.4.2 Sample Configuration for Nessie Catalog and ADLS abfss:// Scheme

```
gg.target=iceberg
gg.eventhandler.iceberg.warehouseLocation=/path/to/iceberg/tables
gg.classpath=DependencyDownloader/dependencies/iceberg-hadoop-azure/
*:DependencyDownloader/dependencies/iceberg-common/*
gg.eventhandler.iceberg.catalogType=nessie
gg.eventhandler.iceberg.nessieBranch=main
gg.eventhandler.iceberg.nessieUri=http://<nessie-server>:10001/api/v2
gg.eventhandler.iceberg.fileSystemScheme=abfss://
gg.eventhandler.iceberg.azureAccountName=<azure-storage-account-name>
gg.eventhandler.iceberg.azureContainer=<azure-storage-container>
gg.eventhandler.iceberg.azureAccountKey=<azure-storage-account-key>
gg.eventhandler.iceberg.proxyServer=<proxy-server>
gg.eventhandler.iceberg.proxyPort=<proxy-port>
```

10.2.12.2.2.4.3 Nessie Namespace

Nessie namespace is the top-level container for all the tables in the Nessie catalog.

Before starting the Replicat process, it is required to have existing namespaces before creating or writing to tables.

Nessie namespace can be created using the nessie command line program (nessie-cli-
<version>.jar) as follows: create namespace QASOURCE;

The Nessie namespace is mapped to the GoldenGate schema in the MAP statement.

For example: MAP QASOURCE.TCUSTMER, TARGET QASOURCE.TCUSTMER;

10.2.12.2.3 Configuration for Iceberg AWS Glue Catalog

- [Configuration for Iceberg AWS Glue Catalog and AWS S3 s3:// OR s3a:// Scheme](#)
- [Classpath and Dependencies](#)
- [Sample Configuration for Iceberg AWS Glue Catalog and AWS S3 s3:// or s3a:// Scheme](#)
- [Table Names and Case Sensitivity](#)

10.2.12.2.3.1 Configuration for Iceberg AWS Glue Catalog and AWS S3 s3:// OR s3a:// Scheme

The following are the configuration properties for the AWS Glue catalog and AWS S3 object store using s3:// or s3a:// scheme:

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .iceberg.catalogType	Optional	String value.	hadoop	glue.
gg.eventhandler .iceberg.awsGlueId	Required	String value.	None	The Glue catalog ID is your numeric AWS account ID.
gg.eventhandler .iceberg.fileSystemScheme	Optional	String value.	file://	File system scheme to indicate AWS S3 object storage location: s3:// or s3a://.
gg.eventhandler .iceberg.awsS3Region	Required	String value.	None	AWS S3 bucket region. Example: us-east-2.
gg.eventhandler .iceberg.awsS3Bucket	Required	String value.	None	AWS S3 bucket name that houses the Iceberg Warehouse.
gg.eventhandler .iceberg.awsAccessKeyId	Optional	String value.	None	AWS access key id for authentication.
gg.eventhandler .iceberg.awsSecretKey	Optional	String value.	None	AWS secret access key for authentication.
gg.eventhandler .iceberg.awsSessionToken	Optional	String value.	None	AWS session token for authentication.
gg.eventhandler .iceberg.awsRoleArn	Optional	String value.	None	AWS role ARN for authentication.
gg.eventhandler .iceberg.awsS3Endpoint	Optional	String value.	None	AWS S3 endpoint.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.proxyServer	Optional	String value.	None	Proxy server to connect to the AWS S3 object storage.
gg.eventhandler.iceberg.proxyPort	Optional	String Value.	80	Proxy server port to connect to the AWS S3 object storage.
gg.eventhandler.iceberg.awsS3PathStyleAccess	Optional	true or false	None	Enables path-style access for S3-compatible storage (for example, domain.com/bucket instead of virtual-hosted style like bucket.domain.com). This is typically required when using non-AWS S3-compatible storage, such as Dell ECS or MinIO.

10.2.12.2.3.2 Classpath and Dependencies

The Java classpath (gg.classpath) should include the following dependencies:

- Iceberg common dependencies
- AWS SDK dependencies for writing to AWS S3 (s3://)

10.2.12.2.3.3 Sample Configuration for Iceberg AWS Glue Catalog and AWS S3 s3:// or s3a:// Scheme

```
gg.target=iceberg
gg.eventhandler.iceberg.warehouseLocation=/path/to/iceberg/tables
gg.classpath=DependencyDownloader/dependencies/iceberg-aws-java-sdk/
*:DependencyDownloader/dependencies/iceberg-common/*
gg.eventhandler.iceberg.catalogType=glue
gg.eventhandler.iceberg.awsGlueId=<aws-account-id>
gg.eventhandler.iceberg.fileSystemScheme=s3://
#gg.eventhandler.iceberg.fileSystemScheme=s3a://
gg.eventhandler.iceberg.awsS3Region=us-east-2
gg.eventhandler.iceberg.awsS3Bucket=<s3-bucket>
gg.eventhandler.iceberg.awsAccessKeyId=<access-key-id>
gg.eventhandler.iceberg.awsSecretKey=<secret-key>
gg.eventhandler.iceberg.proxyServer=<proxy-server>
gg.eventhandler.iceberg.proxyPort=<proxy-port>
```

10.2.12.2.3.4 Table Names and Case Sensitivity

AWS Glue catalog supports only lower case names.

AWS Glue catalog supports only two-part table names.

The target table in the GGDAA Replicat MAP statement should be mapped to the Glue database and table names.

Example: MAP QASOURCE.TCUSTMER, TARGET "glue_database"."tcustmer";

In this example, glue_database is the Glue database name and tcustmer is the Glue table name.

10.2.12.2.4 Configuration for Iceberg Polaris Catalog

Apache Polaris is an open-source, fully-featured catalog for Apache Iceberg.

There are a few options to setup Polaris:

- Snowflake hosted Polaris (<https://other-docs.snowflake.com/en/opencatalog/overview>).
- Polaris on your own infrastructure (<https://polaris.apache.org/in-dev/unreleased/quickstart/>).

Polaris catalog setup includes configuration and authentication to the object stores (S3/GCS/ADLS).

Iceberg warehouse location and authentication to object stores is not setup by GoldenGate when using Polaris.

This topic contains the following:

- [Polaris Common Configuration](#)
- [Polaris Catalog with Google Cloud Storage \(GCS\)](#)
- [Polaris Catalog with AWS S3 Storage](#)
- [Polaris Catalog with Azure Data Lake Storage \(ADLS\)](#)
- [Polaris Catalog and GCS Storage Classpath And Dependencies](#)
- [Polaris Catalog and AWS S3 storage Classpath and Dependencies](#)
- [Polaris Catalog and ADLS storage Classpath And Dependencies](#)
- [Sample Configuration for Polaris Catalog](#)
- [Polaris Namespace](#)

10.2.12.2.4.1 Polaris Common Configuration

The following are the configuration properties for the Polaris catalog:

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.catalogType	Required	String value.	hadoop	polaris.
gg.eventhandler.iceberg.polarisCatalogUri	Required	String value.	None	Polaris Catalog endpoint URI. Example: <code>https://<polaris-account>.snowflakecomputing.com/polaris/api/catalog.</code>

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .iceberg.polarisCatalogName	Required	String value.	None	Polaris Catalog name. Catalog name is the entry point to the Polaris catalog namespace and tables.
gg.eventhandler .iceberg.polarisClientId	Required	String value.	None	Polaris principal's client ID used for authentication and authorization to the respective Polaris catalog.
gg.eventhandler .iceberg.polarisClientSecret	Required	String value.	None	Polaris principal's client secret used for authentication and authorization to the respective Polaris catalog.
gg.eventhandler .iceberg.polarisPrincipalRole	Optional	String value.	ALL	The role to be assumed by the Polaris principal.

10.2.12.2.4.2 Polaris Catalog with Google Cloud Storage (GCS)

The environment variable `GOOGLE_APPLICATION_CREDENTIALS` must be set to the path to the Google Service account key file. Add the following to the Replicat parameter file (.prm):

```
SETENV (GOOGLE_APPLICATION_CREDENTIALS = "/path/to/the/gcp-service-account-  
json-key.json")
```

10.2.12.2.4.3 Polaris Catalog with AWS S3 Storage

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .iceberg.awsS3Region	Required	String value.	None	Required only if the Polaris catalog points to AWS S3 Storage. AWS S3 bucket region. Example: us-east-2.
gg.eventhandler .iceberg.fileSystemScheme	Optional	String value.	file://	Required only if the Polaris catalog points to AWS S3 Storage. File system scheme to indicate AWS S3 object storage location: s3://.
gg.eventhandler .iceberg.awsAccessKeyId	Optional	String value.	None	Required only if the Polaris catalog points to AWS S3 Storage. AWS access key id for authentication.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .iceberg.awsSecretKey	Optional	String value.	None	Required only if the Polaris catalog points to AWS S3 Storage. AWS secret access key for authentication.
gg.eventhandler .iceberg.awsSessionToken	Optional	String value.	None	Required only if the Polaris catalog points to AWS S3 Storage. AWS session token for authentication.
gg.eventhandler .iceberg.awsS3Endpoint	Optional	String value.	None	Required only if the Polaris catalog points to AWS S3 Storage. AWS S3 endpoint.
gg.eventhandler .iceberg.awsS3PathStyleAccess	Optional	true or false	None	Enables path-style access for S3-compatible storage (for example, domain.com/bucket instead of virtual-hosted style like bucket.domain.com). This is typically required when using non-AWS but an S3-compatible storage, such as Dell ECS or MinIO.

10.2.12.2.4.4 Polaris Catalog with Azure Data Lake Storage (ADLS)

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .iceberg.fileSystemScheme	Optional	String value.	file://	Required only if the Polaris catalog points to Azure Data Lake Storage. Warehouse scheme to indicate Azure Data Lake Storage location: abfss://.
gg.eventhandler .iceberg.azureAccountName	Required	String value.	None	Required only if the Polaris catalog points to Azure Data Lake Storage. Azure storage account name that contains the container for the Iceberg Warehouse.

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.iceberg.azureContainer</code>	Required	String value.	None	Required only if the Polaris catalog points to Azure Data Lake Storage. Azure storage account container name that houses the Iceberg Warehouse.
<code>gg.eventhandler.iceberg.azureAccountKey</code>	Required	String value.	None	Required only if the Polaris catalog points to Azure Data Lake Storage. Azure storage account key.
<code>gg.eventhandler.iceberg.azureBlobEndpoint</code>	Optional	String value.	<code><azureContainer>@<azureAccountName>.dfs.core.windows.net</code>	Required only if the Polaris catalog points to Azure Data Lake Storage. Azure Storage service endpoint.
<code>gg.eventhandler.iceberg.azureTenantId</code>	Optional	String value.	None	Azure Tenant ID. Required only for EntraID auth (<code>gg.eventhandler.iceberg.azureAccountKey</code> has to be unset).
<code>gg.eventhandler.iceberg.azureClientId</code>	Optional	String value.	None	Azure EntraID Client ID. Required only for EntraID auth (<code>gg.eventhandler.iceberg.azureAccountKey</code> has to be unset)
<code>gg.eventhandler.iceberg.azureClientSecret</code>	Optional	String value.	None	Azure EntraID client secret. Required only for EntraID auth (<code>gg.eventhandler.iceberg.azureAccountKey</code>) has to be unset)

10.2.12.2.4.5 Polaris Catalog and GCS Storage Classpath And Dependencies

If Polaris catalog is setup to write to GCS, then the Java classpath (`gg.classpath`) should include the following dependencies:

- Iceberg common dependencies
- Google Cloud Storage SDK dependencies for writing to Google Cloud Storage (GCS)

10.2.12.2.4.6 Polaris Catalog and AWS S3 storage Classpath and Dependencies

If Polaris catalog is setup to write to AWS S3, then the Java classpath (gg.classpath) should include the following dependencies:

- Iceberg common dependencies
- AWS SDK dependencies for writing to AWS S3(s3://)

10.2.12.2.4.7 Polaris Catalog and ADLS storage Classpath And Dependencies

If Polaris catalog is setup to write to ADLS, then the Java classpath (gg.classpath) should include the following dependencies:

- Iceberg common dependencies
- Hadoop Azure SDK dependencies for writing to Azure Data Lake Storage (abfss://).

10.2.12.2.4.8 Sample Configuration for Polaris Catalog

```
gg.target=iceberg
#For catalog using GCS
gg.classpath=DependencyDownloader/dependencies/iceberg-gcs-java-sdk/
*:DependencyDownloader/dependencies/iceberg-common/*
#For catalog using S3
#gg.classpath=DependencyDownloader/dependencies/iceberg-aws-java-sdk/
*:DependencyDownloader/dependencies/iceberg-common/*
#For catalog using ADLS
#gg.classpath=DependencyDownloader/dependencies/iceberg-hadoop-azure/
*:DependencyDownloader/dependencies/iceberg-common/*
gg.eventhandler.iceberg.catalogType=polaris
gg.eventhandler.iceberg.polarisCatalogUri=https://<polaris-
account>.snowflakecomputing.com/polaris/api/catalog
gg.eventhandler.iceberg.polarisCatalogName=<polaris_gcs_catalog>
gg.eventhandler.iceberg.polarisClientId=<clientId>
gg.eventhandler.iceberg.polarisClientSecret=<clientSecret>
gg.eventhandler.iceberg.polarisPrincipalRole=ALL
```

10.2.12.2.4.9 Polaris Namespace

Polaris namespace is the top-level container for all the tables in the Polaris catalog.

Before starting the Replicat process, the Polaris namespace should be created in the respective Polaris catalog.

The Polaris namespace is mapped to the GoldenGate schema in the MAP statement.

Example: MAP QASOURCE.TCUSTMER, TARGET "polaris_namespace"."tcustmer";

10.2.12.2.5 Configuration for Iceberg REST Catalog

Iceberg defines a REST specification (<https://github.com/apache/iceberg/blob/main/open-api/rest-catalog-open-api.yaml>) for catalog implementations.

Any REST server that implements the Iceberg REST API can be used as the Iceberg catalog.

For example, Polaris is an implementation of the Iceberg REST API.

- [Configuration for Iceberg REST Catalog](#)
- [Sample Configuration for REST Catalog based on Polaris](#)

- [Sample Rest Catalog Properties file \(For Polaris\)](#)
- [Configuration and Table Names for REST Catalog based on AWS S3 Tables properties](#)
- [Sample configuration for REST Catalog based on IBM watsonx.data](#)
- [Sample Rest Catalog Properties file \(For IBM watsonx.data\)](#)
- [Standalone Utility to Test Connection to Apache Iceberg REST Catalog](#)

10.2.12.2.5.1 Configuration for Iceberg REST Catalog

The following are the configuration properties for the Polaris catalog:

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.catalogType	Required	String value.	hadoop	rest.
gg.eventhandler.iceberg.restCatalogUri	Required	String value.	None	REST Catalog endpoint URI. Example: https://<polaris-account>.snowflakecomputing.com/polaris/api/catalog.
gg.eventhandler.iceberg.restCatalogProperties	Optional	String value.	None	Properties file with additional configuration for the REST catalog.

10.2.12.2.5.2 Sample Configuration for REST Catalog based on Polaris

```
gg.target=iceberg
#For catalog using GCS
gg.classpath=DependencyDownloader/dependencies/iceberg-gcs-java-sdk/
*:DependencyDownloader/dependencies/iceberg-common/*
#For catalog using S3
gg.classpath=DependencyDownloader/dependencies/iceberg-s3/*:DependencyDownloader/
dependencies/iceberg-common/*
#For catalog using ADLS
gg.classpath=DependencyDownloader/dependencies/iceberg-hadoop-azure/
*:DependencyDownloader/dependencies/iceberg-common/*
gg.eventhandler.iceberg.catalogType=rest
gg.eventhandler.iceberg.restCatalogUri=https://<polaris-account>.snowflakecomputing.com/
polaris/api/catalog
gg.eventhandler.iceberg.restCatalogProperties=/path/to/rest/catalog.properties
# Optional configuration for authentication to the object storage.
# Some REST implementations do not require a separate authentication to the storage
layer.
gg.eventhandler.iceberg.fileSystemScheme=s3://
gg.eventhandler.iceberg.awsS3Region=<s3-region>
gg.eventhandler.iceberg.awsS3Bucket=<s3-bucket>
gg.eventhandler.iceberg.awsAccessKeyId=<access-key-id>
gg.eventhandler.iceberg.awsSecretKey=<secret-key>
gg.eventhandler.iceberg.fileSystemScheme=abfs://
gg.eventhandler.iceberg.azureAccountName=<azure-storage-account-name>
gg.eventhandler.iceberg.azureContainer=<azure-storage-container>
gg.eventhandler.iceberg.azureAccountKey=<azure-storage-account-key>
gg.eventhandler.iceberg.fileSystemScheme=gs://
```

```
#gg.eventhandler.iceberg.gcpStorageBucket=<gcs-bucket>
#gg.eventhandler.iceberg.gcpProjectId=<gcp-project-id>
#gg.eventhandler.iceberg.gcpServiceAccountJsonKeyFile=<gcp-service-account-key-file>
```

10.2.12.2.5.3 Sample Rest Catalog Properties file (For Polaris)

```
warehouse=polaris_s3_catalog
credential=<ClientId>:<ClientSecret>
scope=PRINCIPAL_ROLE:ALL
token-refresh-enabled=true
```

10.2.12.2.5.4 Configuration and Table Names for REST Catalog based on AWS S3 Tables properties

```
gg.target=iceberg
gg.classpath=DependencyDownloader/dependencies/iceberg-aws-java-sdk/
*:DependencyDownloader/dependencies/iceberg-common/*
gg.eventhandler.iceberg.catalogType=rest
gg.eventhandler.iceberg.restCatalogUri=https://s3tables.<aws-region>.amazonaws.com/
iceberg
gg.eventhandler.iceberg.restCatalogProperties=/path/to/rest/catalog.properties
gg.eventhandler.iceberg.fileSystemScheme=s3://
gg.eventhandler.iceberg.awsS3Region=<aws-region>
gg.eventhandler.iceberg.awsAccessKeyId=<access-key-id>
gg.eventhandler.iceberg.awsSecretKey=<secret-key>
```

Rest Catalog Properties

```
warehouse=arn:aws:s3tables:<aws-region>:<aws-account-id>:bucket/<s3-table-bucket>
rest.sigv4-enabled=true
rest.signing-name=s3tables
rest.signing-region=<aws-region>
```

- The REST Catalog based on S3 Tables supports only lower case names.
- It supports only two-part table names.
- The target table in the GGDA Replicat MAP statement should be mapped to the S3 Tables namespace and table names. Example, MAP QASOURCE.TCUSTMER, TARGET "s3tables_namespace"."tcustmer". In this example, s3tables_namespace is the S3 Tables namespace name and tcustmer is the table name.
- The S3 Tables namespace must exist before starting the replicat process.
- [AWS Identity and Access Management Permissions for S3 Tables](#)
This section provides a list of the specific Identity and Access Management (IAM) permissions required whenever first-class S3 tables are introduced.

10.2.12.2.5.4.1 AWS Identity and Access Management Permissions for S3 Tables

This section provides a list of the specific Identity and Access Management (IAM) permissions required whenever first-class S3 tables are introduced.

To enable S3 Tables functionality, ensure the following AWS IAM actions are allowed in the policy:

- GetTableBucket
- GetNamespace
- ListNamespaces
- GetTableBucketEncryption

- GetTableBucketMaintenanceConfiguration
- GetTableBucketPolicy
- GetTableMaintenanceConfiguration
- PutTableMaintenanceConfiguration
- PutTablePolicy
- GetTablePolicy
- DeleteTablePolicy
- CreateTable
- GetTable
- GetTableMetadataLocation
- ListTables
- RenameTable
- UpdateTableMetadataLocation
- GetTableData
- PutTableData
- GetTableEncryption
- PutTableEncryption
- DeleteTable

10.2.12.2.5.5 Sample configuration for REST Catalog based on IBM watsonx.data

```

gg.target=iceberg
#For catalog using ADLS
gg.classpath=DependencyDownloader/dependencies/iceberg-hadoop-azure/
*:DependencyDownloader/dependencies/iceberg-common/*
#For catalog using GCS
#gg.classpath=DependencyDownloader/dependencies/iceberg-gcs-java-sdk/
*:DependencyDownloader/dependencies/iceberg-common/*
#For catalog using S3
#gg.classpath=DependencyDownloader/dependencies/iceberg-s3/*:DependencyDownloader/
dependencies/iceberg-common/*
gg.eventhandler.iceberg.catalogType=rest
gg.eventhandler.iceberg.restCatalogUri=https://<watsonx-uri>/mds/iceberg
gg.eventhandler.iceberg.restCatalogProperties=/path/to/rest/watsonx-data-
catalog.properties
# Optional configuration for authentication to the object storage.
gg.eventhandler.iceberg.fileSystemScheme=abfss://
gg.eventhandler.iceberg.azureAccountName=<azure-storage-account-name>
gg.eventhandler.iceberg.azureContainer=<azure-storage-container>
gg.eventhandler.iceberg.azureAccountKey=<azure-storage-account-key>
#gg.eventhandler.iceberg.fileSystemScheme=s3://
#gg.eventhandler.iceberg.awsS3Region=<s3-region>
#gg.eventhandler.iceberg.awsS3Bucket=<s3-bucket>
#gg.eventhandler.iceberg.awsAccessKeyId=<access-key-id>
#gg.eventhandler.iceberg.awsSecretKey=<secret-key>
#gg.eventhandler.iceberg.fileSystemScheme=gs://
#gg.eventhandler.iceberg.gcpStorageBucket=<gcs-bucket>
#gg.eventhandler.iceberg.gcpProjectId=<gcp-project-id>
#gg.eventhandler.iceberg.gcpServiceAccountJsonKeyFile=<gcp-service-account-key-file>

```

10.2.12.2.5.6 Sample Rest Catalog Properties file (For IBM watsonx.data)

```
warehouse=<wxd-catalog-name> # Replace with actual values
header.Authorization=Basic <Base64 encoded <username:password>> # Replace with actual
values
```

10.2.12.2.5.7 Standalone Utility to Test Connection to Apache Iceberg REST Catalog

Oracle GoldenGate for DAA is compatible with any Apache Iceberg REST catalog implementation. For custom REST catalog implementations, Oracle recommends to test the REST catalog connectivity before configuration of the Replicat group.

Note

Oracle has a standalone utility to investigate catalog connection issues. You can request access to this utility by using a service request.

To install the standalone utility perform the following steps:

1. Download the standalone program `ggicebergcatalogutil-<version>.jar`.
2. Ensure that you have Java in the class path; Java run time version should be ≥ 11 .
3. Run the program as follows:

```
java -jar ggicebergcatalogutil-<version>.jar
```

The following message gets displayed:

```
Usage: Provide two arguments: <namespace> '</path/to/iceberg-rest-
catalog.properties>'
```

Sample property file for Polaris REST catalog

```
warehouse=<polaris-catalog-name> # Replace with actual polaris catalog name
uri=https://<polaris-account>.snowflakecomputing.com/polaris/api/catalog #
Replace with actual uri
credential=<polaris-client-id>:<polaris-client-secret> # Replace with
actual credentials
scope=PRINCIPAL_ROLE:ALL # Replace with actual scope
token-refresh-enabled=true
```

Sample property file for IBM Watsonx REST catalog

```
uri=https://<watsonx-uri>/mds/iceberg # Replace with actual uri
warehouse=<wxd-catalog-name> # Replace with actual values
header.Authorization=Basic <Base64 encoded <username:password>> # Replace
with actual values
```

Sample property file for any custom REST catalog

```
# Add properties required for the custom REST Catalog
```

4. Create a catalog properties file for the REST catalog and run the following command:

```
java -jar ggicebergcatalogutil-<version>.jar <namespace> '/path/to/iceberg-catalog.properties'
```

5. If the REST catalog properties and the namespace names are valid and the connection is successful, then the output should list tables in the namespace.

Example

```
Tables in namespace {QASOURCE}:
- {QASOURCE.P_S3_SV_TCUSTOMER} - {QASOURCE.P_S3_SV_TCUSTORD} -
{QASOURCE.TCUSTOMER} - {QASOURCE.TCUSTORD}
```

10.2.12.2.6 Configuration for Iceberg Hadoop Catalog

Hadoop catalog is not recommended for production usage as it has no reliable locking mechanism and would impact concurrent reads and writes.

Hadoop catalog is used for testing purposes only.

- [Configuration for Iceberg Hadoop Catalog and file:// Scheme](#)
- [Configuration for Iceberg Hadoop Catalog and s3a:// Scheme](#)
- [Configuration for Iceberg Hadoop Catalog and gs:// Scheme](#)
- [Configuration for Iceberg Hadoop Catalog and abfss:// Scheme](#)

10.2.12.2.6.1 Configuration for Iceberg Hadoop Catalog and file:// Scheme

The following are the configuration properties for the Hadoop catalog and the local file system as the Iceberg storage using file:// scheme:

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .iceberg.catalogType	Optional	String value.	hadoop	hadoop.
gg.eventhandler .iceberg.fileSystemScheme	Optional	String value.	file://	File system scheme to indicate local file system as the storage: file://.
gg.eventhandler .iceberg.warehouseLocation	Required	String value.	None	Local directory path to the Iceberg warehouse.

Note

This configuration is typically used for testing purposes for storing the Iceberg tables on the local file system.

- [Classpath and Dependencies](#)
- [Sample Configuration for Iceberg Hadoop Catalog and Local File Storage file:// Scheme](#)

10.2.12.2.6.1.1 Classpath and Dependencies

The Java classpath (gg.classpath) should include the following dependencies:

- Iceberg common dependencies

10.2.12.2.6.1.2 Sample Configuration for Iceberg Hadoop Catalog and Local File Storage file:// Scheme

```
gg.target=iceberg
gg.eventhandler.iceberg.warehouseLocation=/path/to/iceberg/tables
gg.classpath=DependencyDownloader/dependencies/iceberg-common/*
```

10.2.12.2.6.2 Configuration for Iceberg Hadoop Catalog and s3a:// Scheme

The following are the configuration properties for the Hadoop catalog and AWS S3 object store using s3a:// scheme:

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.catalogType	Optional	String value.	hadoop	hadoop.
gg.eventhandler.iceberg.fileSystemScheme	Optional	String value.	file://	File system scheme to indicate AWS S3 object storage location: s3a://.
gg.eventhandler.iceberg.awsS3Bucket	Required	String value.	None	AWS S3 bucket name that houses the Iceberg Warehouse.
gg.eventhandler.iceberg.awsAccessKeyId	Required	String value.	None	AWS access key id for authentication.
gg.eventhandler.iceberg.awsSecretKey	Required	String value.	None	AWS secret access key for authentication.
gg.eventhandler.iceberg.awsSessionToken	Optional	String value.	None	AWS session token for authentication.
gg.eventhandler.iceberg.proxyServer	Optional	String value.	None	Proxy server to connect to the AWS S3 object storage.
gg.eventhandler.iceberg.proxyPort	Optional	String value.	80	Proxy server port to connect to the AWS S3 object storage.
gg.eventhandler.iceberg.awsS3Endpoint	Optional	Any valid URL string.	None	The AWS S3 endpoint URL. This is typically required when using non-AWS S3-compatible storage, such as Dell ECS or MinIO.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.awsS3PathStyleAccess	Optional	true or false	None	Enables path-style access for S3-compatible storage (for example, domain.com/bucket instead of virtual-hosted style like bucket.domain.com). This is typically required when using non-AWS S3-compatible storage, such as Dell ECS or MinIO.

- [Classpath and Dependencies](#)
- [Sample Configuration for Hadoop Catalog and AWS S3 s3a:// Scheme](#)

10.2.12.2.6.2.1 Classpath and Dependencies

The Java classpath (gg.classpath) should include the following dependencies:

- Iceberg common dependencies
- Hadoop AWS SDK dependencies for writing to AWS S3 (s3a:// scheme)

10.2.12.2.6.2.2 Sample Configuration for Hadoop Catalog and AWS S3 s3a:// Scheme

```
gg.target=iceberg
gg.eventhandler.iceberg.warehouseLocation=/path/to/iceberg/tables
gg.classpath=DependencyDownloader/dependencies/iceberg-hadoop-aws/*:DependencyDownloader/dependencies/iceberg-common/
gg.eventhandler.iceberg.catalogType=hadoop
gg.eventhandler.iceberg.fileSystemScheme=s3a://
gg.eventhandler.iceberg.awsS3Region=us-east-2
gg.eventhandler.iceberg.awsS3Bucket=<s3-bucket>
gg.eventhandler.iceberg.awsAccessKeyId=<access-key-id>
gg.eventhandler.iceberg.awsSecretKey=<secret-key>
gg.eventhandler.iceberg.proxyServer=<proxy-server>
gg.eventhandler.iceberg.proxyPort=<proxy-port>
```

10.2.12.2.6.3 Configuration for Iceberg Hadoop Catalog and gs:// Scheme

The following are the configuration properties for the Hadoop catalog and GCS object store using gs:// scheme:

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.catalogType	Optional	String value.	hadoop	hadoop.
gg.eventhandler.iceberg.fileSystemScheme	Optional	String value.	file://	File system scheme to indicate GCS object storage location: gs://.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.gcpStorageBucket	Required	String value.	None	Google Cloud Storage bucket name that houses the Iceberg Warehouse.
gg.eventhandler.iceberg.gcpProjectId	Required	String value.	None	Sets the project-id of the Google Cloud project that houses the GCS bucket.
gg.eventhandler.iceberg.gcpServiceAccountJsonKeyFile	Required	String value.	None	Sets the path to the Google Service account key file.
gg.eventhandler.iceberg.proxyServer	Optional	String value.	None	Proxy server to connect to the GCS object storage.
gg.eventhandler.iceberg.proxyPort	Optional	String value.	80	Proxy server port to connect to the GCS object storage.

- [Classpath and Dependencies](#)
- [Sample Configuration for Hadoop Catalog and GCS gs:// Scheme](#)

10.2.12.2.6.3.1 Classpath and Dependencies

The Java classpath (gg.classpath) should include the following dependencies:

- Iceberg common dependencies
- Hadoop Google Cloud Storage SDK dependencies for writing to Google Cloud Storage (GCS)

10.2.12.2.6.3.2 Sample Configuration for Hadoop Catalog and GCS gs:// Scheme

```
gg.target=iceberg
gg.eventhandler.iceberg.warehouseLocation=/path/to/iceberg/tables
gg.classpath=DependencyDownloader/dependencies/iceberg-hadoop-gcs/*:DependencyDownloader/dependencies/iceberg-common/*
gg.eventhandler.iceberg.catalogType=hadoop
gg.eventhandler.iceberg.fileSystemScheme=gs://
gg.eventhandler.iceberg.gcpStorageBucket=<gcs-bucket>
gg.eventhandler.iceberg.gcpProjectId=<gcp-project-id>
gg.eventhandler.iceberg.gcpServiceAccountJsonKeyFile=<gcp-service-account-key-file>
gg.eventhandler.iceberg.proxyServer=<proxy-server>
gg.eventhandler.iceberg.proxyPort=<proxy-port>
```

10.2.12.2.6.4 Configuration for Iceberg Hadoop Catalog and abfss:// Scheme

The following are the configuration properties for the Hadoop catalog and Azure Data Lake Storage using abfss:// scheme:

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.iceberg.catalogType	Optional	String value	hadoop	hadoop.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .iceberg.fileSystemScheme	Optional	String value	file://	File system scheme to indicate Azure Data Lake Storage location: abfss://.
gg.eventhandler .iceberg.azureAccountName	Required	String value	None	Azure storage account name that contains the container for the Iceberg Warehouse.
gg.eventhandler .iceberg.azureContainer	Required	String value	None	Azure storage account container name that houses the Iceberg Warehouse.
gg.eventhandler .iceberg.azureAccountKey	Required	String value.	None	Azure storage account key.
gg.eventhandler .iceberg.azureBlobEndpoint	Optional	String value.	\	Azure Storage service endpoint.
gg.eventhandler .iceberg.proxyServer	Optional	String value.	None	Proxy server to connect to the Azure object storage.
gg.eventhandler .iceberg.proxyPort	Optional	String value.	80	Proxy server port to connect to the Azure object storage.
gg.eventhandler .iceberg.azureTenantId	Optional	String value.	None	Azure Tenant ID. Required only for EntraID auth (gg.eventhandler.iceberg.azureAccountKey has to be unset).
gg.eventhandler .iceberg.azureClientId	Optional	String value.	None	Azure EntraID Client ID. Required only for EntraID auth (gg.eventhandler.iceberg.azureAccountKey has to be unset)
gg.eventhandler .iceberg.azureClientSecret	Optional	String value.	None	Azure EntraID client secret. Required only for EntraID auth (gg.eventhandler.iceberg.azureAccountKey) has to be unset)

- [Classpath and Dependencies](#)

- [Sample Configuration for Hadoop Catalog and ADLS abfss:// Scheme](#)

10.2.12.2.6.4.1 Classpath and Dependencies

The Java classpath (gg.classpath) should include the following dependencies:

- Iceberg common dependencies
- Hadoop Azure SDK dependencies for writing to Azure Data Lake (ADLS)

10.2.12.2.6.4.2 Sample Configuration for Hadoop Catalog and ADLS abfss:// Scheme

```
gg.target=iceberg
gg.eventhandler.iceberg.warehouseLocation=/path/to/iceberg/tables
gg.classpath=DependencyDownloader/dependencies/iceberg-hadoop-azure/
*:DependencyDownloader/dependencies/iceberg-common/*
gg.eventhandler.iceberg.catalogType=hadoop
gg.eventhandler.iceberg.fileSystemScheme=abfss://
gg.eventhandler.iceberg.azureAccountName=<azure-storage-account-name>
gg.eventhandler.iceberg.azureContainer=<azure-storage-container>
gg.eventhandler.iceberg.azureAccountKey=<azure-storage-account-key>
gg.eventhandler.iceberg.proxyServer=<proxy-server>
gg.eventhandler.iceberg.proxyPort=<proxy-port>
```

10.2.12.3 Configuration Templates

Iceberg configuration templates are available in the directory /path/to/AdapterExamples/bigdata/iceberg.

The following template properties files are packaged with Oracle GoldenGate:

- iceberg-glue-s3.properties
- iceberg-hadoop-adls.properties
- iceberg-hadoop-gcs.properties
- iceberg-hadoop-localfile.properties
- iceberg-hadoop-s3.properties
- iceberg-nessie-adls.properties
- iceberg-nessie-gcs.properties
- iceberg-nessie-s3.properties
- iceberg-nessie-s3a.properties
- iceberg-polaris-adls.properties
- iceberg-polaris-gcs.properties
- iceberg-polaris-s3.properties
- iceberg-rest.properties

10.2.12.4 Limitations

- Oracle GoldenGate does not support configuration of partition columns during automatic table creation.
If partitioned tables are required, the Iceberg table should be created manually with the required partition columns.
- Altering the partitioning schema of a table is not supported after starting the Replication process.

If the partitioning schema of a table needs to be changed, the table should be dropped and recreated manually in the target database.

The data in the table will need to be reloaded.

Note

Contact Oracle Support for assistance with this process.

- Pre-existing Iceberg target tables must have identifier columns(key columns) in the schema.
The Replicat process will ABEND if the target table does not have identifier columns.
- The following Iceberg data types cannot be used as a key column (Iceberg identifier field):
 - binary
 - fixed
 - uuid

10.2.12.5 Instantiating Oracle GoldenGate with an Initial Load

For more information about the standard steps for instantiation, see: <https://docs.oracle.com/en/middleware/goldengate/core/21.3/admin/instantiating-oracle-goldengate-initial-load.html#GUID-7D3BD34D-490B-4E76-A48B-63572D93881A>

- [Instantiation Steps Specific to Iceberg](#)
- [Iceberg Change Synchronization Replicat Behavior During Instantiation](#)
- [Configuration for Third-Party S3-Compatible Storage](#)

10.2.12.5.1 Instantiation Steps Specific to Iceberg

1. Start initial load groups for Extract and Replicat.
2. Start change synchronization group for Extract and write operations to a trail file.

Note

Do not start change synchronization group for Replicat yet.

3. Wait until the initial load Replicat group has completed apply of the initial load trail files.
4. Stop the change synchronization group for Extract.
5. Configure a change synchronization Replicat group.
6. Add the parameter UPDATEINSERTS to the change synchronization Replicat group.
7. Start the change synchronization Replicat group.
8. Wait until the change synchronization Replicat group has processed all the trails generated by change synchronization Extract group.
The last record's end offset in the last trail file must match the targetCheckpoint value in the JSON checkpoint file of the change synchronization Replicat group.

Example:

- Run `ls -l` on the last trail file.

```
-rw-r--r-- 1 username dba 5660 Feb 22 2024 /path/to/trail/tr000000003
```

- Here the last record's end offset is 5660, and the trail sequence is 3.
- Open JSON checkpoint file for the change synchronization Replicat group. This should have the following attribute:

```
"targetCheckpoint" : {  
  "trailSequence" : 3,  
  "trailOffset" : 5660  
}
```

This `targetCheckpoint` must match the last record's end offset.

9. Shutdown change synchronization Replicat group and remove the parameter `UPDATEINSERTS`.
10. Initial load is complete now. Start change synchronization Extract and Replicat groups.

10.2.12.5.2 Iceberg Change Synchronization Replicat Behavior During Instantiation

- Execute `[DELETE+INSERT]` for all the `INSERT` operations, irrespective of whether the base row exists on the target or not.
- Run `[DELETE+INSERT]` for all the `UPDATE` operations, irrespective of whether the base row exists on the target or not.
- Run `DELETE` for all the `DELETE` operations, irrespective of whether the base row exists on the target or not.

Note

No collisions will be logged in the Iceberg Replicat report file.

10.2.12.5.3 Configuration for Third-Party S3-Compatible Storage

Third-party S3-compatible storage refers to object storage systems that emulate the AWS S3 API but are provided by vendors other than AWS, such as Dell ECS or MinIO.

These can be used as the backing storage for Iceberg tables in Oracle GoldenGate, often in on-premises, hybrid, or custom cloud environments. To access third-party S3-compatible storage, you typically need to configure the `gg.eventhandler.iceberg.awsS3Endpoint` property to point to your storage endpoint (for example, a custom URL such as, `https://your-ecs-host:9021`). Additionally, enable path-style access by setting `gg.eventhandler.iceberg.awsS3PathStyleAccess` to `true`, as virtual-hosted style (for example, `bucket.domain.com`) is often not supported by non-AWS systems. These properties can be used with various Iceberg catalogs (for example, Hadoop, Nessie, or REST) and integrate with authentication mechanisms like access keys.

Note

This configuration is typically required for non-AWS storage to avoid DNS resolution issues (for example, `UnknownHostException`). Test connectivity before deployment, and ensure your storage system fully supports the S3 API for Iceberg operations.

- [Configuration Properties for Third-Party S3-Compatible Storage](#)

10.2.12.5.3.1 Configuration Properties for Third-Party S3-Compatible Storage

The following properties are used to configure access to third-party S3-compatible storage. These can be combined with catalog-specific settings:

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.iceberg.awsS3Endpoint</code>	Optional / Required (for non-AWS)	Any valid URL string	None	The endpoint URL for the S3-compatible storage (for example, <code>https://your-ecs-host:9021</code> for Dell ECS or <code>http://localhost:9000</code> for MinIO). This overrides the default AWS S3 endpoint.
<code>gg.eventhandler.iceberg.awsS3PathStyleAccess</code>	Optional/ (typically required)	true or false	None	Enables path-style access (for example, <code>domain.com/bucket/path</code> instead of virtual-hosted style like <code>bucket.domain.com</code>). Set to true for most third-party S3-compatible systems, as they often do not support virtual-hosted style.
<code>gg.eventhandler.iceberg.fileSystemScheme</code>	Optional	String value.	None	File system scheme for S3-compatible storage (typically <code>s3://</code> or <code>s3a://</code>). Use <code>s3a://</code> for Hadoop-compatible access.
<code>gg.eventhandler.iceberg.awsAccessKeyId</code>	Optional	String value.	None	Access key ID for authentication to the S3-compatible storage.
<code>gg.eventhandler.iceberg.awsSecretKey</code>	Optional	String value.	None	Secret access key for authentication to the S3-compatible storage.

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.iceberg.awsS3Region</code>	Optional	String value.	None	Region for the S3-compatible storage (if required by the provider; for example, <code>us-east-1</code> for compatibility). Not always needed for third-party systems.
<code>gg.eventhandler.iceberg.proxyServer</code>	Optional	String value.	None	Proxy server hostname or IP for connecting to the S3-compatible storage.
<code>gg.eventhandler.iceberg.proxyPort</code>	Optional	String value.	80	Proxy server port for connecting to the S3-compatible storage.

10.2.12.6 Iceberg Version 3 Support

Iceberg Version 3 introduces enhancements, such as an improved type system that better supports complex data. Enable Iceberg Version 3 by setting the following configuration property:

```
gg.eventhandler.iceberg.icebergVersion=3
```

Note

Before you migrate to Version 3, verify that your query engine is compatible with the underlying Iceberg library. You cannot downgrade from Version 3 to an earlier version; this upgrade is one-way.

- [New Datatypes Supported in V3](#)
- [Limitations](#)

10.2.12.6.1 New Datatypes Supported in V3

Variant

Oracle GoldenGate trail file columns that contain JSON or XML values map to the Iceberg variant type. Oracle GoldenGate identifies:

- JSON columns in the trail file with sub data type value 15.
- XML columns in the trail file with sub data type value 10.

Note

The variant type is not supported when the Iceberg file format is set to ORC, for example, `gg.eventhandler.iceberg.fileFormat=orc`. This limitation exists because ORC does not support complex types in the Iceberg Version 3 Java SDK.

10.2.12.6.2 Limitations

When you use Iceberg Version 3 with Oracle GoldenGate, note the following limitations:

- Iceberg Version 3-specific data types such as `timestamp_ns`, `timestamp_tz_ns`, `geometry`, and `geography` are not supported.
- You can upgrade an existing Iceberg table from Version 2 to Version 3, but you must plan and test the upgrade carefully. For upgrade procedures, refer to the *Apache Iceberg documentation*.

10.2.12.7 Troubleshooting and Diagnostics

- Oracle GoldenGate replicat supports the Iceberg data types as per the version 2 specification.
- Iceberg identifier(key) fields cannot be null. Therefore, the Replicat process will ABEND if the key column value is null.
- Schema changes to the table such as ADD/ALTER/DROP columns is not supported while Replicat process is running.
There are steps to quiesce the replication process, apply the schema changes and resume the replication process.

Note

Contact Oracle Support for assistance with this process.

The Replicat process will ABEND if there are unmapped columns in the target table.

- Replicat ABEND with the following message:

```
ICEBERGEH-00060 Operation record at position '00000000030000003318' for the table
'hadoop.oggdb1.types_tab' has missing column values in an UPDATE. Replicat will
ABEND. To override this behavior set
'gg.eventhandler.iceberg.abendOnMissingColumns=false' and restart the Replicat
process. Setting this property to false will instruct Replicat to
lookup missing columns from the target table and therefore may impact performance.
```

By default, the Iceberg Replicat process expects trails files without missing column value in the UPDATE operations. Replicat can be configured to process compressed trails files with missing column values in the UPDATE operations by setting the property `gg.eventhandler.iceberg.abendOnMissingColumns=false`.

- Replicat ABEND with the following message:

```
ICEBERGEH-00057 Detected changes in the partition columns for the table
'hadoop.oggdb1.types_tab'.
Partition columns in the previous run: '<column list>', partition columns in this
run: '<column list>'.
GoldenGate does not support changing partition columns.
```

Alter the table manually to match the partition columns in the previous run and restart the replicat process.

The Iceberg Replicat process does not support changing partition columns.

- Replicat ABEND with the following message:

ICEBERGEH-00067 Invalid state. The column '<column_name>' in the target table '<table_name>' is not mapped.

The following are the mapped columns: '<column list>'. Iceberg Replicat requires all the columns in the target table to be mapped.

Please map the column '<unmapped column>' and restart the Replicat process.

The Iceberg Replicat process requires all the columns in the target table to be mapped.

- Replicat ABEND with the following message:

ICEBERGEH-00068 Key column '<column name>' in the table '<table name>' is of type float or double.

Iceberg does not support float or double type as identifier (key) fields. Initiating Replicat process shutdown.

Please modify the table schema to exclude double/float types as key columns and restart the Replicat process.

As per the current Iceberg specification (version 2), the column types double and float cannot be used as identifier (key) columns.

- Replicat ABEND with the following message:

ICEBERGEH-00070 Table '<table_name>' contains a key column '<column_name>' of '<binary/fixed/uuid>' type that is not supported by GoldenGate.

The following column types are not supported as key: 'binary, fixed, uuid'. To proceed, either use a supported Iceberg key column type by altering the 'KEYCOLS' clause in the Replicat 'MAP' statement as per the following example: 'MAP <sourceSchema>.<sourceTable>, TARGET <targetSchema>.<targetTable>, KEYCOLS("key1", "key2");' or alter the Iceberg target tables's identifier fields to exclude the key column types that are not supported by GoldenGate.

You can use the following Iceberg SQL statement to alter the table schema: 'ALTER TABLE prod.db.sample SET IDENTIFIER FIELDS key1, key2'.

The Iceberg types binary, fixed and uuid cannot be used as identifier (key) columns.

- Replicat ABEND with the following message:

ICEBERGEH-00071=Table '<table_name>' does not define an Iceberg identifier column. Identifier columns are used as key columns by GoldenGate. Initiating Replicat process shutdown.

Please alter the Iceberg target tables's schema to add identifier columns.

You can use the following Iceberg SQL statement to alter the table schema: 'ALTER TABLE prod.db.sample SET IDENTIFIER FIELDS key1, key2'.

The Iceberg target table should have identifier columns (key columns) in the schema.

- Exceptions in the Replicat handler log file:

- com.google.cloud.storage.StorageException: 401 Unauthorized

- org.apache.iceberg.exceptions.RuntimeIOException: Failed to get file system for path

- org.apache.iceberg.exceptions.RuntimeIOException: Failed to create file

- org.apache.iceberg.exceptions.ForbiddenException: Forbidden

These are common exceptions due to the incorrect configuration of the object storage authentication properties.

Ensure that the following properties are set:

- * `gg.eventhandler.iceberg.fileSystemScheme,`
`gg.eventhandler.iceberg.proxyServer, gg.eventhandler.iceberg.proxyPort`
- * `gg.eventhandler.iceberg.awsAccessKeyId,`
`gg.eventhandler.iceberg.awsSecretKey,`
`gg.eventhandler.iceberg.awsS3Region`
- * `gg.eventhandler.iceberg.azureAccountKey.`

Note

You can use only `gg.eventhandler.iceberg.azureAccountKey` or use `gg.eventhandler.iceberg.azureTenantId,`
`gg.eventhandler.iceberg.azureClientId,`
`gg.eventhandler.iceberg.azureClientSecret` for object storage authentication.

- * `gg.eventhandler.iceberg.gcpProjectId,`
`gg.eventhandler.iceberg.gcpServiceAccountJsonKeyFile.`
- Connectivity to any custom Apache Iceberg REST catalog can be investigated using a standalone utility. For more information, see [Standalone Utility to Test Connection to Apache Iceberg REST Catalog](#).

Connectivity to third-party S3 compatible Storage:

- **Typical Configuration Needs:** For Dell ECS or MinIO, always set `gg.eventhandler.iceberg.awsS3Endpoint` to your custom endpoint and configure `gg.eventhandler.iceberg.awsS3PathStyleAccess` to avoid compatibility issues with virtual-hosted style access.
- `java.net.UnknownHostException: getFileStatus on s3a://<s3-bucket-name>/some/path/metadata/v0.metadata.json:`
`software.amazon.awssdk.core.exception.SdkClientException: Received an UnknownHostException when attempting to interact with a service:`

This appears to be a DNS resolution issue for the URL `<s3-bucket-name>.<s3-endpoint>`, but contributing factors may vary.

Check the following Issue

- Dell ECS not supporting virtual-hosted style access.
Enable Path-Style Access: Set
`gg.eventhandler.iceberg.awsS3PathStyleAccess=true.`

Specify your custom endpoint with `gg.eventhandler.iceberg.awsS3Endpoint` (for example, `http://<s3-endpoint>:9020` for Dell ECS).
- Verify DNS Resolution Manually. Test the full hostname:
Ping `<s3-bucket-name>.<s3-endpoint>`, `nslookup <s3-bucket-name>.<s3-endpoint>` If it fails (for example, `unknown host`), but ping `<s3-bucket-name>.<s3-endpoint>` succeeds, it is a DNS issue.

Confirm with your ECS team if virtual-hosted style is supported.
- Use AWS CLI for S3-Compatible Testing. Test access to the specific bucket using path-style syntax: `aws --endpoint-url http://<s3-endpoint> s3 ls s3://<s3-bucket>/.`

10.2.13 Azure Blob Storage

Topics:

- [Overview](#)
- [Prerequisites](#)
- [Storage Account, Container, and Objects](#)
- [Configuration](#)
- [Troubleshooting and Diagnostics](#)

10.2.13.1 Overview

Azure Blob Storage (ABS) is a service for storing objects in Azure cloud. It is highly scalable and is a secure object storage for cloud-native workloads, archives, data lakes, high-performance computing, and machine learning. You can use the Azure Blob Storage Event handler to load files generated by the File Writer handler into ABS.

10.2.13.2 Prerequisites

Ensure that the following are set:

- Azure cloud account set up.
- Java Software Development Kit (SDK) for Azure Blob Storage.

10.2.13.3 Storage Account, Container, and Objects

- **Storage Account:** An Azure storage account contains all of your Azure Storage data objects: blobs, file shares, queues, tables, and disks.
- **Container:** A container organizes a set of blobs, similar to a directory in a file system. A storage account can include an unlimited number of containers, and a container can store an unlimited number of blobs.
- **Objects/blobs:** Objects or blobs are the individual pieces of data that you store in a storage account container.

10.2.13.4 Configuration

To enable the selection of the ABS Event Handler, you must first configure the Event Handler type by specifying `gg.eventhandler.name.type=abs` and the following ABS properties:

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.type</code>	Required	abs	None	Selects the ABS Event Handler for use with File Writer handler.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .name.bucketMap pingTemplate	Required	A string with resolvable keywords and constants used to dynamically generate a Azure storage account container name.	None	A container is created by the ABS Event handler if it does not exist using this name. See https://docs.microsoft.com/en-us/rest/api/storageservices/naming-and-referencing-containers--blobs--and-metadata#container-names . For supported keywords, see Template Keywords
gg.eventhandler .name.pathMappingTemplate	Required	A string with resolvable keywords and constants used to dynamically generate the path in the Azure storage account container to write the file.	None	Use keywords interlaced with constants to dynamically generate a unique Azure storage account container path names at runtime. Sample path name: ogg/data/\${groupName}/\${fullyQualifiedTableName}. For supported keywords, see Template Keywords
gg.eventhandler .name.fileNameMappingTemplate	Optional	A string with resolvable keywords and constants used to dynamically generate a file name for the Azure Blob object.	None	Use resolvable keywords and constants used to dynamically generate the Azure Blob object file name. If not set, the upstream file name is used. For supported keywords, see Template Keywords
gg.eventhandler .name.finalizeAction	Optional	none delete	none	Set to none to leave the Azure Blob data file in place on the finalize action. Set to delete if you want to delete the Azure Blob data file with the finalize action.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler. .name.eventHandler	Optional	A unique string identifier cross referencing a child event handler.	No event handler configured.	Sets the downstream event handler that is invoked on the file roll event.
gg.eventhandler. .name.accountName	Required	String	None	Azure storage account name.
gg.eventhandler. .name.accountKey	Optional	String	None	Azure storage account key.
gg.eventhandler. .name.sasToken	Optional	String	None	Sets a credential that uses a shared access signature (SAS) to authenticate to an Azure Service.
gg.eventhandler. .name.tenantId	Optional	String	None	Sets the Azure tenant ID of the application.
gg.eventhandler. .name.clientId	Optional	String	None	Sets the Azure client ID of the application.
gg.eventhandler. .name.clientSecret	Optional	String	None	Sets the Azure client secret for the authentication.
gg.eventhandler. .name.accessTier	Optional	Hot Cool Archive	None	Sets the tier on a Azure blob/object. Azure storage offers different access tiers, allowing you to store blob object data in the most cost-effective manner. Available access tiers include Hot, Cool and Archive. For more information, see https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blob-storage-tiers .
gg.eventhandler. .name.endpoint	Optional	String	<a href="https://<accountName>.blob.core.windows.net">https://<accountName>.blob.core.windows.net	Sets the Azure Storage service endpoint. See Azure Government Cloud Configuration

- [Classpath Configuration](#)
- [Dependencies](#)

- [Authentication](#)
- [Proxy Configuration](#)
- [Sample Configuration](#)
- [Azure Government Cloud Configuration](#)

10.2.13.4.1 Classpath Configuration

The ABS Event handler uses the Java SDK for Azure Blob Storage.

Note

Ensure that the classpath includes the path to the Azure Blob Storage Java SDK.

10.2.13.4.2 Dependencies

Download the SDK using the following maven co-ordinates:

```
<dependencies>
  <dependency>
    <groupId>com.azure</groupId>
    <artifactId>azure-storage-blob</artifactId>
    <version>12.13.0</version>
  </dependency>
  <dependency>
    <groupId>com.azure</groupId>
    <artifactId>azure-identity</artifactId>
    <version>1.3.3</version>
  </dependency>
</dependencies>
```

10.2.13.4.3 Authentication

You can authenticate the Azure Storage device by configuring one of the following:

- accountKey
- sasToken
- tenantId, clientId, and clientSecret

accountKey has the highest precedence, followed by sasToken. If accountKey and sasToken are not set, then the tuple tenantId, clientId, and clientSecret are used.

- [Azure Tenant ID, Client ID, and Client Secret](#)

10.2.13.4.3.1 Azure Tenant ID, Client ID, and Client Secret

You can authenticate the Azure Storage device by configuring one of the following:

To obtain your Azure tenant ID:

1. Go to the Microsoft Azure portal.
2. Select Azure Active Directory from the list on the left to view the Azure Active Directory panel.
3. Select Properties in the Azure Active Directory panel to view the Azure Active Directory properties.

The Azure tenant ID is the field marked as Directory ID.

To obtain your Azure client ID and client secret:

1. Go to the Microsoft Azure portal.
2. Select **All Services** from the list on the left to view the Azure Services Listing.
3. Enter **App** into the filter command box and select **App Registrations** from the listed services.
4. Select the App Registration you created to access Azure Storage.

The Application Id displayed for the App Registration is the client ID. The client secret is the generated key string when a new key is added. This generated key string is available only once when the key is created. If you do not know the generated key string, then create another key making sure you capture the generated key string.

10.2.13.4.4 Proxy Configuration

When the process is run behind a proxy server, the `jvm.bootoptions` property can be used to set proxy server configuration using well-known Java proxy properties.

For example:

```
jvm.bootoptions=-Dhttps.proxyHost=some-proxy-address.com -Dhttps.proxyPort=80
-Djava.net.useSystemProxies=true
```

10.2.13.4.5 Sample Configuration

```
#The ABS Event Handler
gg.eventhandler.abs.type=abs
gg.eventhandler.abs.pathMappingTemplate=${fullyQualifiedTableName}
#TODO: Edit the Azure Blob Storage container name
gg.eventhandler.abs.bucketMappingTemplate=<abs-container-name>
gg.eventhandler.abs.finalizeAction=none
#TODO: Edit the Azure storage account name.
gg.eventhandler.abs.accountName=<storage-account-name>
#TODO: Edit the Azure storage account key.
#gg.eventhandler.abs.accountKey=<storage-account-key>
#TODO: Edit the Azure shared access signature(SAS) to authenticate to an Azure
Service.
#gg.eventhandler.abs.sasToken=<sas-token>
#TODO: Edit the the tenant ID of the application.
gg.eventhandler.abs.tenantId=<azure-tenant-id>
#TODO: Edit the the client ID of the application.
gg.eventhandler.abs.clientId=<azure-client-id>
#TODO: Edit the the client secret for the authentication.
gg.eventhandler.abs.clientSecret=<azure-client-secret>
gg.classpath=/path/to/abs-deps/*
#TODO: Edit the proxy configuration.
#jvm.bootoptions=-Dhttps.proxyHost=some-proxy-address.com -Dhttps.proxyPort=80 -
Djava.net.useSystemProxies=true
```

10.2.13.4.6 Azure Government Cloud Configuration

Additional configuration is required if Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) has to replicate data to storage accounts that reside in Azure Government cloud.

Set the environment variables `AZURE_AUTHORITY_HOST` and `gg.eventhandler.{name}.endpoint` as per the following table:

Government cloud	AZURE_AUTHORITY_HOST	gg.eventhandler. {name}.endpoint
Azure US Government Cloud	https:// login.microsoftonline.us.	https://<storage-account- name>.blob.core.usgovcloud api.net
Azure German Cloud	https:// login.microsoftonline.de	https://<storage-account- name>.blob.core.cloudapi.d e
Azure China Cloud	https:// login.chinacloudapi.cn	https://<storage-account- name>.blob.core.chinacloud api.cn

The environment variable can be set in the replicat prm file using the Oracle GoldenGate setenv parameter.

Example:

```
setenv (AZURE_AUTHORITY_HOST = "https://login.microsoftonline.us")
```

10.2.13.5 Troubleshooting and Diagnostics

Error: Confidential Client is not supported in Cross Cloud request.

This indicates that the target Azure storage account resides in one of the Azure Government clouds. Set the required configuration as per [Azure Government Cloud Configuration](#).

Duplicate records after Replicat Recovery

ADLS replication uses File Writer Handler and ADLS Handler in the replicat. Oracle GoldenGate prioritizes no data loss and guarantees no data loss in case of failures by at least once semantics in ADLS (json, csv, delimitedtext, avro_orc, parquet) delivery. In the cases if replicat runs fine and normally shut down, then exactly once is supported. In case of failures (because of network failures), there are various reason that can lead into duplicates in recovery.

Two cases where duplicates can occur

1. If data is written and a failure occurs between when the data is written, and when the checkpoint is moved. Then upon restart the replicat backs up to the previous checkpoint and data can unfortunately be replayed.
2. The rolling of the data files occurs based on customer configured triggers. Trigger can be file size, time, inactivity, or time of day. The rolling does not necessarily happen on a transaction commit boundary. The trigger causes writing to the current file to complete, the post processing transformation and movement complete, and any state on that file is deleted. If a replicat abend occurs in between when the rolling is processed and when the checkpoint is moved, then upon restart, it can again replay those messages.

If you observe duplicate records in case of ADLS replicat recovery, then it is an expected behavior. If you observe duplicates while replicat is running fine, then file a support ticket.

10.2.14 Azure Data Lake Storage

- [Azure Data Lake Gen1 \(ADLS Gen1\)](#)
Microsoft Azure Data Lake supports streaming data through the Hadoop client. Therefore, data files can be sent to Azure Data Lake using either the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) Hadoop Distributed File System (HDFS) Handler or the File Writer Handler in conjunction with the HDFS Event Handler.
- [Azure Data Lake Gen2 using Hadoop Client and ABFS](#)
Microsoft Azure Data Lake Gen 2 (using Hadoop Client and ABFS) supports streaming data via the Hadoop client. Therefore, data files can be sent to Azure Data Lake Gen 2 using either the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) HDFS Handler or the File Writer Handler in conjunction with the HDFS Event Handler.
- [Azure Data Lake Gen2 using BLOB endpoint](#)

10.2.14.1 Azure Data Lake Gen1 (ADLS Gen1)

Microsoft Azure Data Lake supports streaming data through the Hadoop client. Therefore, data files can be sent to Azure Data Lake using either the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) Hadoop Distributed File System (HDFS) Handler or the File Writer Handler in conjunction with the HDFS Event Handler.

The preferred mechanism for ingest to Microsoft Azure Data Lake is the File Writer Handler in conjunction with the HDFS Event Handler.

Use these steps to connect to Microsoft Azure Data Lake from GG for DAA.

1. Download Hadoop 2.9.1 from <http://hadoop.apache.org/releases.html>.
2. Unzip the file in a temporary directory. For example, `/ggwork/hadoop/hadoop-2.9`.
3. Edit the `/ggwork/hadoop/hadoop-2.9/hadoop-env.sh` file in the directory.
4. Add entries for the `JAVA_HOME` and `HADOOP_CLASSPATH` environment variables:

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
export HADOOP_CLASSPATH=/ggwork/hadoop/hadoop-2.9.1/share/hadoop/tools/lib/
*:$HADOOP_CLASSPATH
```

This points to Java 8 and adds the `share/hadoop/tools/lib` to the Hadoop classpath. The library path is not in the variable by default and the required Azure libraries are in this directory.

5. Edit the `/ggwork/hadoop/hadoop-2.9.1/etc/hadoop/core-site.xml` file and add:

```
<configuration>
<property>
<name>fs.adl.oauth2.access.token.provider.type</name>
<value>ClientCredential</value>
</property>
<property>
<name>fs.adl.oauth2.refresh.url</name>
<value>Insert the Azure https URL here to obtain the access token</value>
</property>
<property>
<name>fs.adl.oauth2.client.id</name>
<value>Insert the client id here</value>
</property>
</property>
```



```

<name>fs.adl.oauth2.credential</name>
<value>Insert the password here</value>
</property>
<property>
<name>fs.defaultFS</name>
<value>adl://Account Name.azuredatalakestore.net</value>
</property>
</configuration>

```

6. Open your firewall to connect to both the Azure URL to get the token and the Azure Data Lake URL. Or disconnect from your network or VPN. Access to Azure Data Lake does not currently support using a proxy server per the Apache Hadoop documentation.
7. Use the Hadoop shell commands to prove connectivity to Azure Data Lake. For example, in the 2.9.1 Hadoop installation directory, execute this command to get a listing of the root HDFS directory.

```
./bin/hadoop fs -ls /
```

8. Verify connectivity to Azure Data Lake.
9. Configure either the HDFS Handler or the File Writer Handler using the HDFS Event Handler to push data to Azure Data Lake, see [Flat Files](#). Oracle recommends that you use the File Writer Handler with the HDFS Event Handler.

Setting the `gg.classpath` example:

```

gg.classpath=/ggwork/hadoop/hadoop-2.9.1/share/hadoop/common/:/ggwork/hadoop/hadoop-
2.9.1/share/hadoop/common/lib/:/ggwork/hadoop/hadoop-
2.9.1/share/hadoop/hdfs/:/ggwork/hadoop/hadoop-2.9.1/share/hadoop/hdfs/lib/:/ggwork/
hadoop/hadoop-
2.9.1/etc/hadoop:/ggwork/hadoop/hadoop-2.9.1/share/hadoop/tools/lib/*

```

See <https://hadoop.apache.org/docs/current/hadoop-azure-datalake/index.html>.

10.2.14.2 Azure Data Lake Gen2 using Hadoop Client and ABFS

Microsoft Azure Data Lake Gen 2 (using Hadoop Client and ABFS) supports streaming data via the Hadoop client. Therefore, data files can be sent to Azure Data Lake Gen 2 using either the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) HDFS Handler or the File Writer Handler in conjunction with the HDFS Event Handler.

Hadoop 3.3.0 (or higher) is recommended for connectivity to Azure Data Lake Gen 2. Hadoop 3.3.0 contains an important fix to correctly fire Azure events on file close using the "abfs" scheme. For more information, see [Hadoop Jira issue Hadoop-16182](#).

Use the File Writer Handler in conjunction with the HDFS Event Handler. This is the preferred mechanism for ingest to Azure Data Lake Gen 2.

Prerequisites

Part 1:

1. Connectivity to Azure Data Lake Gen 2 assumes that you have correctly provisioned an Azure Data Lake Gen 2 account in the Azure portal. From the Azure portal select **Storage Accounts** from the commands on the left to view/create/delete storage accounts.

In the Azure Data Lake Gen 2 provisioning process, it is recommended that the Hierarchical namespace is enabled in the **Advanced** tab.

It is not mandatory to enable Hierarchical namespace for Azure storage account.

2. Ensure that you have created a Web app/API App Registration to connect to the storage account.
From the Azure portal select All services from the list of commands on the left, type app into the filter command box and select App registrations from the filtered list of services. Create an App registration of type Web app/API.

Add permissions to access Azure Storage. Assign the App registration to an Azure account. Generate a Key for the App Registration as follows:

- a. Navigate to the respective App registration page.
- b. On the left pane, select **Certificates & secrets**.
- c. Click **+ New client secret** (This should show a new key under the column **Value**).

The generated key string is your client secret and is only available at the time the key is created. Therefore, ensure you document the generated key string.

Part 2:

1. In the Azure Data Lake Gen 2 account, ensure that the App Registration is given access. In the **Azure** portal, select **Storage accounts** from the left panel. Select the Azure Data Lake Gen 2 account that you have created.

Select the **Access Control (IAM)** command to bring up the **Access Control (IAM)** panel. Select the **Role Assignments** tab and add a role assignment for the created App Registration.

The app registration assigned to the storage account must be provided with read and write access into the Azure storage account.

You can use either of the following roles: the built-in Azure role Storage Blob Data Contributor or custom role with the required permissions.

2. Connectivity to Azure Data Lake Gen 2 can be routed through a proxy server. Three parameters need to be set in the Java boot options to enable:

```
jvm.bootoptions=-Xmx512m -Xms32m -Djava.class.path=ggjava/ggjava.jar -DproxySet=true
-Dhttps.proxyHost={insert your proxy server} -Dhttps.proxyPort={insert your proxy
port}
```

3. Two connectivity schemes to Azure Data Lake Gen 2 are supported: abfs and abfss. The preferred method is abfss since it employs HTTPS calls thereby providing security and payload encryption.

Connecting to Microsoft Azure Data Lake 2

To connect to Microsoft Azure Data Lake 2 from GG for DAA:

1. Download Hadoop 3.3.0 from <http://hadoop.apache.org/releases.html>.
2. Unzip the file in a temporary directory. For example, /usr/home/hadoop/hadoop-3.3.0.
3. Edit the {hadoop install dir}/etc/hadoop/hadoop-env.sh file to point to Java 8 and add the Azure Hadoop libraries to the Hadoop classpath. These are entries in the hadoop-env.sh file:

```
export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_202
export HADOOP_OPTIONAL_TOOLS="hadoop-azure"
```

4. Private networks often require routing through a proxy server to access the public internet. Therefore, you may have to configure proxy server settings for the hadoop command line utility to test the connectivity to Azure. To configure proxy server settings, set the following in the hadoop-env.sh file:

```
export HADOOP_CLIENT_OPTS="-Dhttps.proxyHost={insert your proxy server} -
Dhttps.proxyPort={insert your proxy port}"
```

Note

These proxy settings only work for the hadoop command line utility. The proxy server settings for GG for DAA connectivity to Azure are set in the `jvm.bootoptions` as described in this point.

5. Edit the `{hadoop install dir}/etc/hadoop/core-site.xml` file and add the following configuration:

```
<configuration>
<property>
  <name>fs.azure.account.auth.type</name>
  <value>OAuth</value>
</property>
<property>
  <name>fs.azure.account.oauth.provider.type</name>
  <value>org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider</value>
</property>
<property>
  <name>fs.azure.account.oauth2.client.endpoint</name>
  <value>https://login.microsoftonline.com/{insert the Azure Tenant id here}/oauth2/
token</value>
</property>
<property>
  <name>fs.azure.account.oauth2.client.id</name>
  <value>{insert your client id here}</value>
</property>
<property>
  <name>fs.azure.account.oauth2.client.secret</name>
  <value>{insert your client secret here}</value>
</property>
<property>
  <name>fs.defaultFS</name>
  <value>abfss://{insert your container name here}@{insert your ADL gen2 storage
account name here}.dfs.core.windows.net</value>
</property>
<property>
  <name>fs.azure.createRemoteFileSystemDuringInitialization</name>
  <value>true</value>
</property>
</configuration>
```

To obtain your Azure Tenant Id, go to the **Microsoft Azure** portal. Enter Azure Active Directory in the **Search** bar and select **Azure Active Directory** from the list of services. The Tenant Id is located in the center of the main **Azure Active Directory** service page.

To obtain your Azure Client Id and Client Secret go to the Microsoft Azure portal. Select **All Services** from the list on the left to view the Azure Services Listing. Type **App** into the filter command box and select **App Registrations** from the listed services. Select the App Registration that you have created to access Azure Storage. The Application Id displayed for the App Registration is the Client ID. The Client Secret is the generated key string when a new key is added. This generated key string is available only once when the key is created. If you do not know the generated key string, create another key making sure you capture the generated key string.

The ADL gen2 account name is the account name you generated when you created the Azure ADL gen2 account.

File systems are sub partitions within an Azure Data Lake Gen 2 storage account. You can create and access new file systems on the fly but only if the following Hadoop configuration is set:

```
<property>
  <name>fs.azure.createRemoteFileSystemDuringInitialization</name>
  <value>true</value>
</property>
```

6. Verify connectivity using Hadoop shell commands.

```
./bin/hadoop fs -ls /
./bin/hadoop fs -mkdir /tmp
```

7. Configure either the HDFS Handler or the File Writer Handler using the HDFS Event Handler to push data to Azure Data Lake, see [Flat Files](#). Oracle recommends that you use the File Writer Handler with the HDFS Event Handler.

Setting the `gg.classpath` example:

```
gg.classpath=/ggwork/hadoop/hadoop-3.3.0/share/hadoop/common/*:/ggwork/hadoop/
hadoop-3.3.0/share/hadoop/common/lib/*:/ggwork/hadoop/hadoop-3.3.0/share/hadoop/
hdfs/*:
/ggwork/hadoop/hadoop-3.3.0/share/hadoop/hdfs/lib/*:/ggwork/hadoop/hadoop-3.3.0/etc/
hadoop:/ggwork/hadoop/hadoop-3.3.0/share/hadoop/tools/lib/*
```

See <https://hadoop.apache.org/docs/current/hadoop-azure-datalake/index.html>.

10.2.14.3 Azure Data Lake Gen2 using BLOB endpoint

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) can connect to ADLS Gen2 using BLOB endpoint. GG for DAA ADLS Gen2 replication using BLOB endpoint does not require any Hadoop installation. For more information, see [Azure Blob Storage](#).

10.2.15 Azure Event Hubs

Kafka handler supports connectivity to Microsoft Azure Event Hubs.

To connect to the Microsoft Azure Event Hubs:

1. For more information about connecting to Microsoft Azure Event Hubs, see [Quickstart: Data streaming with Event Hubs using the Kafka protocol](#).
2. Update the Kafka Producer Configuration file as follows to connect to Microsoft Azure Event Hubs using Secure Sockets Layer (SSL)/Transport Layer Security (TLS) protocols:

```
bootstrap.servers=NAMESPACENAME.servicebus.windows.net:9093
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule
required username="$ConnectionString"
password="{YOUR.EVENTHUBS.CONNECTION.STRING}";
```

See [Kafka Producer Configuration File](#).

Connectivity to the Azure Event Hubs cannot be routed through a proxy server. Therefore, when you run Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) on premise to push data to Azure Event Hubs, you need to open your firewall to allow connectivity.

10.2.16 Azure Synapse Analytics Data Warehouse

Microsoft Azure Synapse Analytics is a limitless analytics service that brings together data integration, enterprise data warehousing and Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) analytics.

- [Detailed Functionality](#)
- [Operation Aggregation](#)
- [Compressed Update Handling](#)
- [Configuration](#)
- [Restrictions on Database Object Names](#)
- [Troubleshooting and Diagnostics](#)

10.2.16.1 Detailed Functionality

Replication to Synapse uses stage and merge data flow.

The change data is staged in a temporary location in micro-batches and eventually merged into the target table.

Azure Data Lake Storage (ADLS) Gen 2 is used as the staging area for change data.

The Synapse Event handler is used as a downstream Event handler connected to the output of the Parquet Event handler.

The Parquet Event handler loads files generated by the File Writer Handler into ADLS Gen2.

The Synapse Event handler executes SQL statements to merge the operation records staged in ADLS Gen2.

The SQL operations are performed in batches providing better throughput.

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) uses the MERGE SQL statement or a combination of DELETE and INSERT SQL statements to perform the merge operation.

- [Database User Privileges](#)
- [Merge SQL Statement](#)
- [Prerequisites](#)

10.2.16.1.1 Database User Privileges

Database user used for replication has to be granted the following privileges:

- INSERT, UPDATE, DELETE, and TRUNCATE on the target tables.
- CREATE and DROP Synapse external file format.
- CREATE and DROP Synapse external data source.
- CREATE and DROP Synapse external table.
- Optionally, CREATE Synapse target table, if setting `gg.handler.synapse.createTable` to true.

10.2.16.1.2 Merge SQL Statement

The merge SQL statement for Azure Synapse Analytics was made generally available during the later part of the year 2022 and therefore Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) uses merge statement by default. To disable merge SQL, ensure that a Java System property is set in the `jvm.bootoptions` parameter.

For example:

```
jvm.bootoptions=-Dsynapse.use.merge.sql=false
```

10.2.16.1.3 Prerequisites

The following are the prerequisites:

- Uncompressed UPDATE records: If Oracle GoldenGate is configured to not use merge statement (see [Merge SQL Statement](#)), then it is mandatory that the trail files used to apply to Synapse contain uncompressed UPDATE operation records, which means that the UPDATE operations contain full image of the row being updated. If UPDATE records have missing columns, then replicat will ABEND on detecting a compressed UPDATE trail record.
- If Oracle GoldenGate is configured to use merge statement (see [Merge SQL Statement](#)), then the target table must be a hash distributed table.
- Azure storage account: An Azure storage account and container should exist. Oracle recommends co-locating the Azure Synapse workspace, and the Azure storage account in the same azure region.
- If Oracle GoldenGate is configured to use merge statement, then the target table cannot define IDENTITY columns because Synapse merge statement does not support inserting data into IDENTITY columns. For more information about merging SQL statement, see [Merge SQL Statement](#).

10.2.16.2 Operation Aggregation

Operation aggregation is the process of aggregating (merging/compressing) multiple operations on the same row into a single output operation based on a threshold.

- [In-Memory Operation Aggregation](#)
- [Operation Aggregation Using SQL](#)

10.2.16.2.1 In-Memory Operation Aggregation

- Operation records are aggregated in-memory by default.
- The `gg.aggregate.operations.flush.interval` property has been deprecated and is no longer supported. If `gg.aggregate.operations.flush.interval` is used in GG for DAA 23ai, then replicat will run; but add a warning to log file about the property being deprecated and not supported.
To control the time window for aggregation, use the `gg.handler.synapse.fileRollInterval` property. By default, it is set to 3 minutes. Longer intervals will increase latency, and may increase memory usage. Shorter intervals will increase overhead in Oracle GoldenGate and the target database.
- Operation aggregation in-memory requires additional JVM memory configuration.

10.2.16.2.2 Operation Aggregation Using SQL

- To use SQL aggregation, it is mandatory that the trail files contain uncompressed UPDATE operation records, which means that the UPDATE operations contain full image of the row being updated.
- Operation aggregation using SQL can provide better throughput if the trails files contains uncompressed update records.
- Replicat can aggregate operations using SQL statements by setting the `gg.aggregate.operations.using.sql=true`.
- You can tune the frequency of merge interval using the File writer `gg.handler.synapse.fileRollInterval` property, the default value is set to 3m (three minutes).
- Operation aggregation using SQL does not require additional JVM memory configuration.

10.2.16.3 Compressed Update Handling

A compressed update record contains values for the key columns and the modified columns.

An uncompressed update record contains values for all the columns.

Oracle GoldenGate trails may contain compressed or uncompressed update records. The default extract configuration writes compressed updates to the trails. The parameter `gg.compressed.update` can be set to true or false to indicate compressed or uncompressed update records.

The default extract configuration writes compressed updates to the trails. The parameter `gg.compressed.update` can be set to true or false to indicate compressed/uncompressed update records.

- [MERGE statement with Uncompressed Updates](#)

10.2.16.3.1 MERGE statement with Uncompressed Updates

In some use cases, if the trail contains uncompressed update records, then the MERGE SQL statement can be optimized for better performance by setting `gg.compressed.update=false`.

If you want to use DELETE+INSERT SQL statements instead of a MERGE SQL statement, then set `gg.eventhandler.synapse.deleteInsert=true`.

10.2.16.4 Configuration

- [Automatic Configuration](#)
- [Synapse Database Credentials](#)
- [Classpath Configuration](#)
- [INSERTALLRECORDS Support](#)
- [Large Object \(LOB\) Performance](#)
- [End-to-End Configuration](#)
- [Authenticate using EntraID](#)

10.2.16.4.1 Automatic Configuration

Synapse replication requires the configuration of multiple components, including the File Writer handler, Parquet Event handler, Azure Blob Event handler, and Synapse Event handler.

The Automatic Configuration feature simplifies this process by automatically configuring these components, minimizing the required user configuration. Any properties modified during auto-configuration are also logged in the handler log file.

To enable auto-configuration for replication to a Synapse target, set the parameter `gg.target=synapse`.

When replicating to a Synapse target, customization of the Parquet Event handler name and Synapse Event handler name is not supported.

- [File Writer Handler Configuration](#)
- [Parquet Event Handler Configuration](#)
- [Azure Blob Storage Event Handler Configuration](#)
- [Synapse Event Handler Configuration](#)
- [Synapse Auto Creation Target Table Configuration](#)

10.2.16.4.1.1 File Writer Handler Configuration

File writer handler name is pre-set to the value `synapse`. The following is an example to edit a property of File Writer handler:

```
gg.handler.synapse.pathMappingTemplate=./dirout
```

10.2.16.4.1.2 Parquet Event Handler Configuration

Parquet Event Handler name is pre-set to the value `parquet`. The Parquet Event Handler is auto-configured to write to the local file system. The hadoop configuration file `core-site.xml` is not required to be configured to write data files to the local file system.

The following is an example to edit a property of Parquet Event handler:

```
gg.eventhandler.parquet.finalizeAction=delete
```

10.2.16.4.1.3 Azure Blob Storage Event Handler Configuration

The Azure Blob Storage (ABS) Event Handler name is pre-set to the value `abs` and must be configured to match your ADLS Gen2 configuration. The following is an example of editing a property of the ABS Event Handler:

```
gg.eventhandler.abs.bucketMappingTemplate=container1
```

Related Topics

- [Azure Blob Storage](#)

10.2.16.4.1.4 Synapse Event Handler Configuration

Synapse Event Handler name is pre-set to the value `synapse`.

Table 10-13 Synapse Event Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler.synapse.connectionURL	Required	jdbc:sqlserver://<synapse-workspace>.sql.azure-synapse.net:1433;database=<db-name>;encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.sql.azure-synapse.net;loginTimeout=300;	None	JDBC URL to connect to Synapse.
gg.eventhandler.synapse.UserName	Required	Database username.	None	Synapse database user in the Synapse workspace. The username has to be qualified with the Synapse workspace name. Example: sqladminuser@synapseworkspace.
gg.eventhandler.synapse.Password	Required	Supported database string.	None	Synapse database password.
gg.eventhandler.synapse.credential	Required	Credential name.	None	Synapse database credential name to access Azure Data Lake Gen2 files. See Synapse Database Credentials for steps to create credential.
gg.eventhandler.synapse.maxConnections	Optional	Integer value	10	Use this parameter to control the number of concurrent JDBC database connections to the target Synapse database.

Table 10-13 (Cont.) Synapse Event Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler.synapse.dropStagingTablesOnShutdown	Optional	true or false	false	If set to true, the temporary staging tables created by GoldenGate will be dropped on replicat graceful stop.
gg.maxInlineLobSize	Optional	Integer Value	16000	This parameter can be used to set the maximum inline size of large object (LOB) columns in bytes. For more information, see Large Object (LOB) Performance .

Table 10-13 (Cont.) Synapse Event Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler. synapse.fil eRollInterv al	Optional	The default unit of measure is milliseconds. You can stipulate ms, s, m, h to signify milliseconds, seconds, minutes, or hours respectively. Examples of legal values include 10000, 10000ms, 10s, 10m, or 1.5h. Values of 0 or less indicate that file rolling on time is turned off.	3m (three minutes)	The parameter determines how often the data will be merged into Synapse. Use with caution, the higher this value is the more data will need to be stored in the memory of the Replicat process.

i Note
Use the parameter to manage the number of rows in the table. The higher the value, the more data will be stored in the memory of the Replicat process.

Table 10-13 (Cont.) Synapse Event Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation

r
e
a
s
i
n
g
i
t
s
d
e
f
a
u
l
t
v
a
l
u
e
(
3
m
)
w
i
l
l
i
n
c
r
e
a
s
e
t
h
e
a
m
o
u
n
t
o
f
d
a
t
a

Table 10-13 (Cont.) Synapse Event Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation

Properties	Required/ Optional	Legal Values	Default	Explanation

Table 10-13 (Cont.) Synapse Event Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation

y
.

 N
o
t
e
S
t
a
r
t
i
n
g
w
i
t
h
t
h
e
2
3
a
i
r
e
l
e
a
s
e
t
h
e
g
g
a
g
g
r
e
g
a
t
e

Table 10-13 (Cont.) Synapse Event Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation

.
o
p
e
r
a
t
i
o
n
s
.
f
l
u
s
h
.
i
n
t
e
r
v
a
l
p
r
o
p
e
r
t
y
i
s
d
e
p
r
e
c
a
t
e
d
a
n
d
n
o
l

Table 10-13 (Cont.) Synapse Event Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation

o
n
g
e
r
s
u
p
p
o
r
t
e
d
·
F
o
r
m
o
r
e
i
n
f
o
r
m
a
t
i
o
n
,
s
e
e
[!n = M e m o r y O p e r a t i o](#)

Table 10-13 (Cont.) Synapse Event Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.compress ed.update	Optional	true or false	true	If set the true, then this indicates that the source trail files contain compressed update operations. If set to true, then the source trail files are expected to contain uncompressed update operations.
gg.eventhan dler.synaps e.connectio nRetryInter valSeconds	Optional	Integer Value	30	Specifies the delay in seconds between connection retry attempts.
gg.eventhan dler.synaps e.connectio nRetries	Optional	Integer Value	3	Specifies the number of times connections to the target data warehouse will be retried.

Table 10-13 (Cont.) Synapse Event Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.validate .keyupdate	Optional	true or false	false	If set to true, Replicat will validate key update operations (optype 115) and correct to normal update if no key values have changed. Compressed key update operations do not qualify for merge.

Table 10-13 (Cont.) Synapse Event Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler.synapse.deleteInsert	Optional	true or false	false	If set to true, Replicat will merge records using SQL DELETE+INSERT statements instead of SQL MERGE statement.

Information
Note
A
p
p
l
i
c
a
b
l
e
o
n
l
y
i
f
g
g
.
c
o
m
p
r
e
s
s
e
d
.
u
p
d
a
t

Table 10-13 (Cont.) Synapse Event Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler.synapse.createTable	Optional	true or false	true	If the value is set to true, then target tables are automatically created if missing.

e
i
s
e
t
o
f
a
l
s
e
.

10.2.16.4.1.5 Synapse Auto Creation Target Table Configuration

The Auto Creation feature, is pre-set to true, this means, that the source tables will be automatically generated on the Synapse target if they do not already exist before replicating the data.

If it is desired for the tables to do not get automatically generated at the same time when the data is being replicated, this configuration can be turned off. The following is an example to edit a property of File Writer handler:

```
gg.eventhandler.synapse.createTable=false
```

If the Merge SQL statement is enabled, HASH will be used as the distribution method for the generated tables, and the first primary key column of each table will be chosen as the HASH distribution key. If the Merge SQL statement is disabled, ROUND_ROBIN method will be used as the distribution method, because this is Synapse's default configuration. For more information, see [Merge SQL Statement](#).

10.2.16.4.2 Synapse Database Credentials

To allow Synapse to access the data files in Azure Data Lake Gen2 storage account, follow the steps to create a database credential:

- 1. Connect to the respective Synapse SQL dedicated pool using the Azure Web SQL console (<https://web.azure.synapse.net/en-us/>).

2. Create a DB master key if one does not already exist, using your own password.
3. Create a database scoped credential. This credential allows Oracle GoldenGate replicat process to access Azure Storage Account.
Provide the Azure Storage Account name and Access key when creating this credential.

Storage Account Access keys can be retrieved from the Azure cloud console.

For example:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Your own password' ;
CREATE DATABASE SCOPED CREDENTIAL OGGBD_ADLS_credential
WITH
-- IDENTITY = '<storage_account_name>' ,
  IDENTITY = 'sanavaccountuseast' ,
-- SECRET = '<storage_account_key>'
  SECRET = 'c8C0yR-this-is-a-fake-access-key-Gc9c5mEN0J1mLyx101vSRDlRG0/
Ke+tbAvi6xe73HAAhLtdMFZRA=='
;
```

10.2.16.4.3 Classpath Configuration

Synapse Event handler relies on the upstream File Writer handler and the Parquet Event handler.

- [Dependencies](#)
- [Classpath](#)

10.2.16.4.3.1 Dependencies

- Microsoft SQLServer JDBC driver: The JDBC driver can be downloaded from Maven central using the following co-ordinates.

```
<dependency>
  <groupId>com.microsoft.sqlserver</groupId>
  <artifactId>mssql-jdbc</artifactId>
  <version>8.4.1.jre8</version>
  <scope>provided</scope>
</dependency>
```

Alternatively, the JDBC driver can also be downloaded using the script <OGGDIR>/DependencyDownloader/synapse.sh.

- Parquet Event handler dependencies: See [Parquet Event Handler Configuration](#) to configure classpath to include Parquet dependencies.
- Hadoop Dependencies: Hadoop dependencies can be downloaded using dependency downloader <OGGDIR>/DependencyDownloader/hadoop.sh
- Azure Storage dependencies: Azure Storage dependencies can be downloaded from Maven using the following co-ordinates.

```
<dependencies>
  <dependency>
    <groupId>com.azure</groupId>
    <artifactId>azure-storage-blob</artifactId>
    <version>12.13.0</version>
  </dependency>
  <dependency>
    <groupId>com.azure</groupId>
    <artifactId>azure-identity</artifactId>
    <version>1.3.3</version>
  </dependency>
</dependencies>
```

```
</dependency>
</dependencies>
```

10.2.16.4.3.2 Classpath

Edit the `gg.classpath` configuration parameter to include the path to the Parquet Event Handler, Synapse JDBC, Azure Storage and hadoop dependencies.

For example:

```
gg.classpath=/path/to/parquet-deps/*:/path/to/abs-deps/*:path/to/synapse-deps/mssql-jdbc-8.4.1.jre8.jar:/path/to/hadoop-deps/*
```

10.2.16.4.4 INSERTALLRECORDS Support

Stage and merge targets supports `INSERTALLRECORDS` parameter.

See [INSERTALLRECORDS](#) in *Reference for Oracle GoldenGate*. Set the `INSERTALLRECORDS` parameter in the Replicat parameter file (`.prm`). Set the `INSERTALLRECORDS` parameter in the Replicat parameter file (`.prm`)

Setting this property directs the Replicat process to use bulk insert operations to load operation data into the target table. You can tune the batch size of bulk inserts using the File Writer property `gg.handler.synapse.maxFileSize`. The default value is set to 1GB. The frequency of bulk inserts can be tuned using the File Writer property `gg.handler.synapse.fileRollInterval`, the default value is set to 3m (three minutes).

① Note

- When using the Synapse internal stage, the staging files can be compressed by setting `gg.handler.synapse.putSQLAutoCompress` to true.

10.2.16.4.5 Large Object (LOB) Performance

The presence of large object (LOB) columns can impact Replicat's apply performance. Any LOB column changes that exceed the inline threshold `gg.maxInlineLobSize` does not qualify for batch processing and such operations gets slower.

If the compute machine has sufficient RAM, you can increase this parameter to speed up processing.

10.2.16.4.6 End-to-End Configuration

The following is an end-end configuration example which uses auto-configuration for FW handler, Parquet and Synapse Event handlers.

This sample properties file can also be found in the directory `AdapterExamples/big-data/synapse/synapse.props`:

```
# Azure Synapse Analytics Data Warehouse Template
# Configuration to load GoldenGate trail operation records into Azure Synapse Analytics
by chaining
# File writer handler -> Parquet Event handler -> Synapse Event handler.
# Note: Recommended to only edit the configuration marked as TODO
```

```
gg.target=synapse
```

```
#The Parquet Event Handler
```

```
No properties are required for the Parquet Event handler.
```

```
#gg.eventhandler.parquet.finalizeAction=delete

ADLS Gen 2 stage (Using Azure Blob SDK).
#Azure Blob Event handler
#TODO: Edit the Azure Blob Storage container name
gg.eventhandler.abs.bucketMappingTemplate=<abs-container-name>
#TODO: Edit the Azure storage.account name. gg.eventhandler.abs.accountName=<storage-
account-name>
#TODO: Edit the Azure storage account key.
gg.eventhandler.abs.accountKey=<storage-account-key>

#The Synapse Event Handler
#TODO: Edit JDBC ConnectionUrl
gg.eventhandler.synapse.connectionURL=jdbc:sqlserver://<synapse-
workspace>.sql.azuresynapse.net:1433;database=<db-
name>;encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.sql.azuresynapse.
net;loginTimeout=300;
#TODO: Edit JDBC user name
gg.eventhandler.synapse.UserName=<db user name>@<synapse-workspace>
#TODO: Edit JDBC password
gg.eventhandler.synapse.Password=<db password>
#TODO: Edit Credential to access Azure storage.
gg.eventhandler.synapse.credential=OGGBD_ADLS_credential
#TODO: Edit the classpath to include dependencies for Parquet Event Handler, ABS Event
handler and the Synapse JDBC driver.
gg.classpath=/path/to/parquet-deps:/path/to/abs-deps:/path/to/synapse-deps/mssql-
jdbc-8.4.1.jre8.jar:/path/to/hadoop-deps/*
#TODO: Provide sufficient memory (at least 8GB).
jvm.bootoptions=-Xmx8g -Xms8g -DSYNAPSE_STAGE=parquet,abs
```

10.2.16.4.7 Authenticate using EntraID

Azure Synapse Analytics supports authentication using Microsoft Entra ID by allowing users to sign-in using their Entra ID credentials.

This can be achieved using the following Oracle GoldenGate Synapse Event Handler properties:

- Set the Entra Client Id instead of username in the `gg.eventhandler.synapse.UserName` property.
- Set the Entra Client Secret instead of password in the `gg.eventhandler.synapse.Password` property.
- Add the following to the `gg.eventhandler.synapse.connectionURL` property:
 - `authentication=ActiveDirectoryServicePrincipal`

The output properties should look like the following:

```
gg.eventhandler.synapse.UserName=00000000-0000-0000-0000-000000000000
gg.eventhandler.synapse.Password=...
gg.eventhandler.synapse.connectionURL=jdbc:sqlserver://
workspace.sql.azuresynapse.net:1433;database=db;
encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.sql.azuresyn
apse.net;loginTimeout=300;authentication=ActiveDirectoryServicePrincipal;
```


10.2.16.5 Restrictions on Database Object Names

- **Catalog, Schema, and Table Names:** Cannot contain period (.), asterisk (*), question mark (?), double quote ("), backslash (\), open bracket ([), close bracket (]), or control characters (ASCII 0-31).
- **Specific limitation for GGDA 21c and previous releases:** Supports only names that comply with the Avro specification for identifier names: must start with `[A-Za-z\_]`, followed by `[A-Za-z0-9\_]`
- Additionally, all object names are also limited by the supported list of characters for the respective target.

10.2.16.6 Troubleshooting and Diagnostics

- **Connectivity Issues to Synapse:**
 - Validate JDBC connection URL, username and password.
 - Check if http/https proxy is enabled. Synapse does not support connections over http(s) proxy.
- **DDL not applied on the target table:** Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) does not support DDL replication.
- **SQL Errors:** In case there are any errors while executing any SQL, the entire SQL statement along with the bind parameter values are logged into the GG for DAA handler log file.
- **Co-existence of the components:** The location/region of the machine where replicat process is running, Azure Data Lake Storage container region and the Synapse region would impact the overall throughput of the apply process. Data flow is as follows: Oracle GoldenGate -> Azure Data Lake Gen 2 -> Synapse. For best throughput, the components need to be located as close as possible.
- **Replicat ABEND due to partial LOB records in the trail file:** GG for DAA Synapse apply does not support replication of partial LOB. The trail file needs to be regenerated by Oracle Integrated capture using `TRANLOGOPTIONS FETCHPARTIALLOB` option in the extract parameter file.
- **Error:com.microsoft.sqlserver.jdbc.SQLServerException: Conversion failed when converting date and/or time from character string:**
This occurs when the source datetime column and target datetime column are incompatible.

For example: A case where the source column is a timestamp type, and the target column is Synapse time.
- If the Synapse table or column names contain double quotes, then GG for DAA replicat will ABEND.
- **Error: com.microsoft.sqlserver.jdbc.SQLServerException: HdfsBridge::recordReaderFillBuffer.** This indicates that the data in the external table backed by Azure Data Lake file is not readable. Contact Oracle support.
- **IDENTITY column in the target table:** The Synapse MERGE statement does not support inserting data into IDENTITY columns. Therefore, if MERGE statement is enabled using `jvm.bootoptions=-Dsynapse.use.merge.sql=true`, then Replicat will ABEND with following error message:

Exception:

```
com.microsoft.sqlserver.jdbc.SQLServerException: Cannot update identity  
column 'ORDER_ID'
```

- **Error: com.microsoft.sqlserver.jdbc.SQLServerException: Merge statements with a WHEN NOT MATCHED [BY TARGET] clause must target a hash distributed table:** This indicates that merge SQL statement is on and Synapse target table is not a hash distributed table. You need to create the target table with a hash distribution.

10.2.17 Databricks

Overview

Databricks is a unified, open analytics platform for building, deploying, sharing, and maintaining enterprise-grade data, analytics, and AI solutions at scale.

- [Detailed Functionality](#)
- [Configuration](#)
- [Troubleshooting and Diagnostics](#)
- [Restrictions on Database Object Names](#)

10.2.17.1 Detailed Functionality

Replication to Databricks uses stage and merge data flow.

The change data from the Oracle GoldenGate trails is staged in micro-batches at a temporary staging location, typically a cloud object store.

The staged records are then merged into the Databricks target tables using a merge SQL statement.

- [Staging location](#)
- [Database User Privileges](#)
- [Prerequisites](#)

10.2.17.1.1 Staging location

The change data records from the GoldenGate trail files are formatted into Avro OCF (Object Container Format) and uploaded to the staging location. Change data can be staged in one of the following object stores based on the Databricks configuration.

- Azure Data Lake Storage (ADLS) Gen2
- AWS Simple Storage Service (S3)
- Google Cloud Storage (GCS)

10.2.17.1.2 Database User Privileges

The database user used for replicating into Databricks has to be granted the following privileges:

- CREATE, INSERT, UPDATE, DELETE, and TRUNCATE on the target tables.
- CREATE, ALTER, and DROP external tables.

10.2.17.1.3 Prerequisites

- You must have Azure, Amazon Web Services, or Google Cloud Platform cloud accounts set up for Databricks.
- Azure storage accounts must have hierarchical namespace enabled for replication to Databricks on Azure.
- Databricks JDBC driver. For more information about the recommended JDBC driver version, see [Databricks documentation](#).

10.2.17.2 Configuration

The configuration of the Databricks replication properties is stored in the Replicat properties file.

- [Automatic Configuration](#)
- [Authentication to Databricks](#)
- [Unity Catalog](#)
- [Create an External Location](#)
- [Compute Clusters Without Unity Catalog](#)
- [Classpath Configuration](#)
- [Proxy Configuration](#)
- [INSERTALLRECORDS Support](#)
- [Operation Aggregation](#)
- [Compressed Update Handling](#)
- [End-to-End Configuration](#)
- [Table Mapping](#)

10.2.17.2.1 Automatic Configuration

Databricks replication involves configuring multiple components, such as the File Writer Handler, ABS or S3 or GCS Event Handler, and the target Databricks Event Handler.

The Automatic Configuration functionality helps you to autoconfigure these components so that the manual configuration is minimal.

The properties modified by autoconfiguration is also logged in the handler log file.

To enable autoconfiguration to replicate to the Databricks target, set the parameter `gg.target=databricks`.

The parameter `gg.stage` determines the staging location.

If `gg.stage` is unset, ADLS Gen2 will be used as the staging location.

If `gg.state` is set to either `abs`, `s3`, or `gcs`, then ADLS Gen2, AWS S3, or GCS are respectively used as the staging locations.

The JDBC Metadata provider is also automatically enabled to retrieve target table metadata from Databricks.

Target tables are automatically created if missing.

- [File Writer Handler Configuration](#)
- [ABS Event Handler Configuration](#)
- [s3 Event Handler Configuration](#)
- [GCS Event Handler Configuration](#)
- [Databricks Event Handler Configuration](#)

10.2.17.2.1.1 File Writer Handler Configuration

The File Writer Handler name is pre-set to the value databricks and its properties are automatically set to the required values for Databricks.

10.2.17.2.1.2 ABS Event Handler Configuration

The ABS Event Handler name is pre-set to the value abs and must be configured to match your ADLS Gen2 configuration.

The following is an example of editing a property of the S3 Event Handler:
`gg.eventhandler.abs.bucketMappingTemplate=container1`

For more information about integrating with ADLS Gen2, see [Azure Blob Storage Configuration](#).

10.2.17.2.1.3 s3 Event Handler Configuration

The s3 Event Handler name is pre-set to the value s3 and must be configured to match your s3 configuration. The following is an example of editing a property of the s3 Event Handler:
`gg.eventhandler.s3.bucketMappingTemplate=bucket1`

For more information about integrating with s3, see [S3 Event handler configuration](#).

10.2.17.2.1.4 GCS Event Handler Configuration

The GCS Event Handler name is pre-set to the value gcs and must be configured to match your GCS configuration. The following is an example of editing a GCS Event Handler property:
`gg.eventhandler.gcs.bucketMappingTemplate=bucket1`

The following is an example of editing a property of the S3 Event Handler:
`gg.eventhandler.abs.bucketMappingTemplate=container1`

For more information about integrating with GCS, see [GCS Event Handler Configuration](#).

10.2.17.2.1.5 Databricks Event Handler Configuration

The Databricks Event Handler name is pre-set to the value databricks.

The following are the configuration properties available for the Databricks Event handler, the required ones must be changed to match your Databricks configuration.

Table 10-14

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler.databricks.connectionURL	Required	jdbc:databricks://<server-hostname>:443;httpPath=<http-path>;<setting1>=<value1>;<setting2>=<value2>;<settingN>=<valueN>]	None	JDBC URL to connect to Databricks. See Databricks Authentication Methods .
gg.eventhandler.databricks.UserName	Optional	Supported database user name string.	None	Databricks database user or token.
gg.eventhandler.databricks.Password	Optional	Supported database password string.	None	Databricks database password or token value.
gg.eventhandler.databricks.credential	Optional	Storage Credential name.	None	External Storage credential name to access files on object storage such as ADLS Gen2, S3 or GCS. For more information, see Create a Storage Credential .
gg.eventhandler.databricks.createTable	Optional	true or false	true	If the value is set to true, then target tables are automatically created if missing.
gg.eventhandler.databricks.maxConnections	Optional	Integer value	10	Use this parameter to control the number of concurrent JDBC database connections to the target database.
gg.eventhandler.databricks.connectionRetries	Optional	Integer value	3	Specifies the number of times connections to the target data warehouse will be retried.
gg.eventhandler.databricks.connectionRetryIntervalSeconds	Optional	Integer value	30	Specifies the delay in seconds between connection retry attempts.

Table 10-14 (Cont.)

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler.databricks.deleteInsert	Optional	true or false	false	If set to true, Replicat will merge records using SQL DELETE+INSERT statements instead of SQL MERGE statement.

Note
Applicable only if gg.compressed.update is

Table 10-14 (Cont.)

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.databricks.detectMissingBaseRow</code>	Optional	true or false	false	Diagnostic parameter to find UPDATE operations without base row. If set to true, Replicat will ABEND if there are UPDATE operations without base row. These rows will be collected into another table that can be investigated.
<code>gg.eventhandler.databricks.dropStagingTablesOnShutdown</code>	Optional	true or false	false	If set to true, the temporary staging tables created by Oracle GoldenGate will be dropped on replicat graceful stop.

Set of all
S
e
t
o
f
a
l
l
S
e

Table 10-14 (Cont.)

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler.data bricks.fileRoll Interval	Optional	The default unit of measure is milliseconds. You can stipulate ms, s, m, h to signify milliseconds, seconds, minutes, or hours respectively. Examples of legal values include 10000, 10000ms, 10s, 10m, or 1.5h. Values of 0 or less indicate that file rolling on time is turned off.	3m (three minutes)	The parameter determines how often the data will be merged into Databricks. Use with caution, the higher this value is the more data will need to be stored in the memory of the Replicat process.

Note
Use the parameter with caution. Increasing

Table 10-14 (Cont.)

Properties	Required/ Optional	Legal Values	Default	Explanation

ing
g
i
t
s
d
e
f
a
u
l
t
v
a
l
u
e
(
3
m
)
w
i
l
l
i
n
c
r
e
a
s
e
t
h
e
a
m
o
u
n
t
o
f
d
a
t
a
s
t
o
r

Table 10-14 (Cont.)

Properties	Required/ Optional	Legal Values	Default	Explanation

ed in the internal memory of the Replicat. This can cause out of m

Table 10-14 (Cont.)

Properties	Required/ Optional	Legal Values	Default	Explanation

memory errors and stop the Replicat if it runs out of memory.

Table 10-14 (Cont.)

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.validate.key update	Optional	true or false	false	If set to true, Replicat will validate key update operations (optype 115) and correct to normal update if no key values have changed. Compressed key update operations do not qualify for merge.
gg.compressed.u pdate	Optional	true or false	true	If set the true, then this indicates that the source trail files contain compressed update operations. If set to false, then the source trail files are expected to contain uncompressed update operations.

 Note

Table 10-14 (Cont.)

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.databricks.deltaUniversalFormat</code>	Optional	<code>iceberg</code> or <code>' '</code> (empty).	<code>' '</code> (empty)	If set to <code>iceberg</code> , Replicat will automatically create target tables with Delta Lake Universal Format set to <code>iceberg</code> . This allows you to read Delta tables with Iceberg reader clients. Universal Format can be enabled only on Databricks tables managed by Unity Catalog. This feature requires Databricks Runtime 14.3 LTS or above. See: https://docs.databricks.com/en/delta/uniform.html
<code>gg.eventhandler.databricks.externalTableRefreshMethod</code>	Optional	<code>REFRESH</code> or <code>ALTER</code> or <code>RECREATE</code>	<code>ALTER</code>	Determines how the external tables used for staging data files are handled. • <code>REFRESH</code> will refresh the external table on each staging event with the new contents of the staging directory. • <code>ALTER</code> will alter the external table to point to the new data file specifically. • <code>RECREATE</code> will drop and create the external table on each staging event.

Table 10-14 (Cont.)

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler.databricks.useExternalTable	Optional	true or false	false	- If set to true, use external staging tables. - If set to false (default), staging data in the data files are accessed without creating an external table using direct path queries. This approach can simplify setup and potentially avoid external table refresh issues by directly querying AVRO files from the storage location, for example, ADLS Gen2, S3, or GCS, within a view. The underlying query might look like: <pre> SELECT `col1`::INT `col1`, `col2`::STRIN G `col2`, ... FROM AVRO. `abfss:// <container>@< storageaccoun t>.dfs.core.w indows.net//o gg/<REP>/ <QUAL_TABLE>/ <REP>_<QUAL_T ABLE_<TIMESTA MP>.dat` </pre> (the exact columns and casts depend on the table metadata).

10.2.17.2.2 Authentication to Databricks

Databricks JDBC connection URLs use the following format: jdbc:databricks://<Host>:<Port>;httpPath=<http-path>[;<setting1>=<value1>;<setting2>=<value2>;<settingN>=<valueN>]

- [Compute Settings for the Databricks JDBC Driver](#)

10.2.17.2.2.1 Compute Settings for the Databricks JDBC Driver

The driver requires the following compute resource configuration settings:

Table 10-15 Compute Settings for the Databricks JDBC Driver

Setting	Description
Host	The Databricks compute resource's Server Hostname value.
Port	443
httpPath	The Databricks compute resource's HTTP Path value.
ssl	1

- [Connection Details for Compute Cluster](#)
- [Connection Details for Databricks SQL Warehouse](#)
- [Databricks Authentication Methods](#)
- [Databricks Personal Access Token](#)
- [Databricks Username and Password](#)
- [Use a Service Principal to Authenticate with Databricks \(OAuth M2M\)](#)

10.2.17.2.2.1.1 Connection Details for Compute Cluster

To get the connection details for the Databricks compute cluster:

- Log in to your Databricks workspace.
- In the sidebar, click **Compute**.
- In the list of available warehouses, click the target cluster's name.
- On the **Configuration** tab, expand **Advanced options**.
- In the **JDBC/ODBC** tab, the **Server hostname**, **Port**, and **HTTP path** can be found.

10.2.17.2.2.1.2 Connection Details for Databricks SQL Warehouse

To get the connection details for the Databricks SQL warehouse:

- Log in to your Databricks workspace.
- In the sidebar, click **SQL Warehouses**.
- In the list of available warehouses, click the target warehouse's name.
- In the **Connection details** tab, the **Server hostname**, **Port**, and **HTTP path** can be found.

10.2.17.2.2.1.3 Databricks Authentication Methods

Only Databricks OAuth machine-to-machine (M2M) authentication is supported.

10.2.17.2.2.1.4 Databricks Personal Access Token

To create a Databricks personal access token:

- In your Databricks workspace, click your Databricks username in the top bar, and then select **User Settings** from the drop down.
- Click **Developer**.
- Next to **Access tokens**, click **Manage**.
- Click **Generate New Token**.
- (Optional) Enter a comment that helps you to identify this token in the future, and change the token's default lifetime of 90 days.
To create a token with no lifetime (not recommended), leave the Lifetime (days) box empty (blank).
- Click **Generate**.
- Copy the displayed token to a secure location, and then click **Done**.

10.2.17.2.2.1.5 Databricks Username and Password

Databricks username and password authentication is also known as Databricks basic authentication. Username and password authentication is possible only if single sign-on is disabled.

To use Databricks username and password for authentication, set the `gg.eventhandler.databricks.UserName` and `gg.eventhandler.databricks.Password` properties to the respective values.

You can choose to configure the JDBC URL as per: <https://docs.databricks.com/en/integrations/jdbc/authentication.html#oauth-20-tokens> . For more information about all the authentication mechanisms supported by Databricks, see <https://docs.databricks.com/en/integrations/jdbc/authentication.html>

10.2.17.2.2.1.6 Use a Service Principal to Authenticate with Databricks (OAuth M2M)

To create a service principal at the workspace level, follow these steps:

1. Login to the Databricks workspace as a workspace admin.
2. Click your username in the top bar of the Databricks workspace and select **Settings**.
3. Click **Identity and Access**.
4. Next to **Service Principals**, click **Manage**.
5. Click **Add service principal**.
6. Click the drop-down arrow in the search box and then click **Add New**.
7. Enter a name for the service principal.
8. Click **Add**.

10.2.17.2.3 Unity Catalog

Unity Catalog provides centralized access control, auditing, lineage, and data discovery capabilities across Databricks workspaces.

In Unity Catalog, the hierarchy of primary data objects flows from metastore to table or volume:

- **Metastore:** The top-level container for metadata. Each metastore exposes a three-level namespace (`catalog.schema.table`) that organizes your data.

Note

If your workspace includes a legacy Hive metastore, then the data in that metastore will still be available alongside data defined in Unity Catalog, in a catalog named `hive_metastore`.

- **Catalog:** The first layer of the object hierarchy, used to organize your data assets.
- **Schema:** Also known as databases, schemas are the second layer of the object hierarchy and contain tables and views.
- **Tables, views, and volumes:** At the lowest level in the data object hierarchy are tables, views, and volumes. Volumes provide governance for non-tabular data.
- [Managed Tables](#)
- [External Tables](#)
- [Create a Storage Credential](#)

10.2.17.2.3.1 Managed Tables

Oracle GoldenGate replicates data to Databricks managed tables. Managed tables use the DELTA table format.

- [Tables Inside hive_metastore Catalog](#)

10.2.17.2.3.1.1 Tables Inside hive_metastore Catalog

If Unity Catalog was enabled on an existing Databricks workspace, then the existing tables are available in the `hive_metastore` catalog.

The tables under the `hive_metastore` do not support primary key.

Oracle GoldenGate Replicat MAP statement should use `KEYCOLS` to define the key columns required for stage/merge replication.

10.2.17.2.3.2 External Tables

External tables are file-backed tables that reference data stored in an external location.

External location is an object storage such as ADLS Gen2, AWS S3, or GCS.

To manage cloud storage for external tables, unity catalog uses the following:

- **Storage Credential:** A storage credential allows Databricks to access data in cloud storage.
- External locations contain a reference to a storage credential and a cloud storage path.

10.2.17.2.3.3 Create a Storage Credential

A storage credential represents an authentication and authorization mechanism for accessing data stored on your cloud tenant.

- [Storage Credential to Access Azure Storage Account](#)
- [Storage Credential to Access Google Storage Account](#)

10.2.17.2.3.3.1 Storage Credential to Access Azure Storage Account

To create a storage credential to access an Azure storage account:

- Create an Azure resource called “Access Connector for Azure Databricks”

- In the Azure portal, search for **Access Connector for Azure Databricks** and select the **Access Connector for Azure Databricks** service.
- Following the steps to create a connector resource.
- In the Azure portal, search for “Storage accounts” and select the storage account that should be used as a GoldenGate staging location.
- In the sidebar, click **Access Control (IAM)**.
- On **Role assignments**, click **+Add**, select **Add role assignment** from the drop-down menu.
- In the search bar, enter **Storage Blob Data Contributor**, select the role from the list, and click **Next**.
- Click **Members: +Select members**, in the search bar, enter the name of the Azure Databricks connector resource, and click **Select**.
- Back in the Azure portal, search for Azure Databricks connector resource, and note down the **Resource ID**.
- Create a storage credential in Databricks:
 - Log in to your Databricks workspace.
 - In the sidebar, click **Catalog**.
 - Click **External Data**, then click **Storage Credentials**.
 - Click **Create Storage Credential**.
 - In the **Create Storage Credential** dialog, enter the following details:
 - * **Credential Type**: Choose **Azure Managed Identity** from the drop-down menu.
 - * **Storage credential name**: Enter a name for the storage credential.
 - * **Access connector ID**: Enter the resource ID of the Azure Databricks connector resource.
 - * Click **Advanced Options**, check the box **Limit to read-only use**.
 - * Click **Create**.

Note

You can create multiple storage credentials for different storage accounts.

10.2.17.2.3.3.2 Storage Credential to Access Google Storage Account

To create a storage credential to access an Google storage account:

- Create a storage credential to access Google Cloud Storage:
 1. Log in to your Databricks workspace.
 2. In the sidebar, click **Catalog**.
 3. Click **External Data**, then click **Storage Credentials**.
 4. Click **Create Storage Credential**.
 5. In the **Create Storage Credential** dialog, enter the following details:
 - **Credential Type**: Choose **Google Service Account** from the drop-down menu.
 - **Storage credential name**: Enter a name for the storage credential.

6. Click **Advanced Options**, check the box **Limit to read-only use**.
 7. Click **Create**.
 8. On the **Storage credential created** dialog, make a note of the service account ID, which is in the form of an email address, and click **Done**.
- Configure permissions for the service account:
 1. Go to the Google Cloud console and open the GCS bucket that you want to access from your Databricks workspace.
The bucket should be in the same region as your Databricks workspace.
 2. On the **Permission** tab, click **+ Grant** access and assign the service account the following roles:
 - Storage Legacy Bucket Reader
 - Storage Object AdminUse the service account's email address as the principal identifier.

Note

You can create multiple storage credentials for different storage accounts.

10.2.17.2.4 Create an External Location

External locations associate Unity Catalog storage credentials with cloud object storage containers.

External locations are used to define managed storage locations for catalogs and schemas, and to define locations for external tables and external volumes.

Oracle GoldenGate recommends creating an external location to simplify configuration, but you may decide to skip this step.

- [External Location to Access Azure Storage Account](#)
- [External Location to Access Google Storage Account](#)

10.2.17.2.4.1 External Location to Access Azure Storage Account

To create an external location:

- Log in to your Databricks workspace.
- In the sidebar, click **Catalog**.
- Click **External Data**, then click on **External Locations**.
- Click **Create Location**.
- In the **Create a new external location** form, enter the following details:
 - **External location name**: Enter a name for the external location.
 - **Storage credential**: Choose the storage credential that you created earlier from the drop-down menu.
 - **URL**: Enter the bucket path that you want to use as the external location. For example:

`abfss://oggcontainer@storageaccount.dfs.core.windows.net/ogg`

- Click **Advanced Options**, check the box **Limit to read-only use**.
- Click **Create**.

① **Note**

If an external location is created and tied to a storage credential, then there is no need to set the event handler property `gg.eventhandler.databricks.credential`.

10.2.17.2.4.2 External Location to Access Google Storage Account

To create an external location:

- Log in to your Databricks workspace.
- In the sidebar, click **Catalog**.
- Click **External Data**, and then click **External Locations**.
- Click **Create Location**.
- In the **Create a new external location** form, enter the following details:
 - **External location name**: Enter a name for the external location.
 - **Storage credential**: Choose the storage credential that you created earlier from the drop-down menu.
 - **URL**: Enter the bucket path that you want to use as the external location. For example:

`gs://gcs-bucket/ogg`
 - Click **Advanced Options**, check the box **Limit to read-only use**.
 - Click **Create**.

10.2.17.2.5 Compute Clusters Without Unity Catalog

Legacy Databricks compute clusters may not have Unity Catalog support, or some compute clusters do not have Unity Catalog enabled.

In these clusters, access to external object storage should be configured by creating Databricks secrets and by setting spark configuration in Databricks compute. Follows one of the configuration steps below depending on your storage service.

Databricks replication without unity catalog also requires modifications in replicat properties:

- In replicat properties, set `useExternalTable` to true:

`gg.eventhandler.databricks.useExternalTable=true`
- In replicat properties, comment `gg.stage=abs`

`#gg.stage=abs`
- [Spark Configuration to Access Azure Storage Account](#)
- [Spark Configuration to Access Google Cloud Storage](#)
- [Spark Configuration to Access AWS S3](#)

- [Creating Databricks Secrets](#)

10.2.17.2.5.1 Spark Configuration to Access Azure Storage Account

To create a Databricks secret, perform the following steps:

1. Log in to your Databricks workspace.
2. From the sidebar, select **Compute**.
3. In the list of available clusters, click the **target cluster's** name.
4. Select **Apps** and then click **Web Terminal**.
A new terminal session opens in a new tab in the web browser
5. Create a Databricks secrets scope using `databricks secrets create-scope <scope-name>`. For example:

```
databricks secrets create-scope gg
```

6. Create a secret using the scope.

```
databricks secrets put-secret --json '{ "scope":
    "<scope-name>", "key": "<key-name>", "string_value":
    "<storage_account_access_key>"
}'
```

For example:

```
databricks secrets put-secret --json '{ "scope": "gg", "key": "mykey",
"string_value":
    "-----storage-account-access-key-----"
}'
```

In Databricks compute, set spark configuration as follows:

1. Log in to the Databricks workspace
2. From the sidebar, select **Compute**.
3. In the list of available clusters, click the **target cluster's** name.
4. Select the **Configuration** tab and then expand the **Advanced options**.
5. In the **Spark** tab, specify the spark configuration for the cluster.

To configure spark configuration for Databricks on Azure:

```
fs.azure.account.key.<your_storage_account_name>.dfs.core.windows.net
{{secrets/<scope_name>/<key_name>}}
```

For example:

```
fs.azure.account.key.<your_storage_account_name>.dfs.core.windows.net
{{secrets/gg/mykey}}
```

10.2.17.2.5.2 Spark Configuration to Access Google Cloud Storage

The following information is required to configure access to Google Cloud Storage:

- Service account email
- Project id
- Service account private key
- Service account private key id

To create a Databricks secret:

1. Log in to your Databricks workspace.
2. From the sidebar, select **Compute**.
3. In the list of available clusters, click the **target cluster's** name.
4. Select **Apps** and then click **Web Terminal**.
A new terminal session opens in a new tab in the web browser
5. Create a Databricks secrets scope using `databricks secrets create-scope <scope-name>`.
For example:

```
databricks secrets create-scope gg
```

6. Store service account private key id using Databricks.

```
secrets put-secret <scope_name> private_key_id --string-value  
<service_account_private_key_id>
```

7. Store service account private key using Databricks.

```
secrets put-secret <scope_name> private_key --string-value "-----BEGIN  
RSA PRIVATE KEY-----\n..."
```

In Databricks compute, set spark configuration as follows:

1. Log in to your Databricks workspace.
2. From the sidebar, select **Compute**.
3. In the list of available clusters, click the **target cluster's** name.
4. On the **Configuration** tab, expand **Advanced options**.
5. In the **Spark** tab, you can specify the spark configuration for the cluster.

To configure spark configuration for Databricks on Google Cloud:

```
google.cloud.auth.service.account.enable true  
fs.gs.auth.service.account.email <client-email>  
fs.gs.project.id <project-id>  
fs.gs.auth.service.account.private.key {{secrets/<scope_name>/  
private_key}}  
fs.gs.auth.service.account.private.key.id {{secrets/<scope_name>/  
private_key_id}}
```

10.2.17.2.5.3 Spark Configuration to Access AWS S3

To create a Databricks secret:

1. Log in to your Databricks workspace.

2. From the sidebar, select **Compute**.
3. In the list of available clusters, click the **target cluster's** name.
4. Select **Apps** and then click **Web Terminal**.
A new terminal session opens in a new tab in the web browser
5. Create a Databricks secrets scope using `databricks secrets create-scope <scope-name>`. For example:

```
databricks secrets create-scope gg
```

6. Store AWS access key id using:

```
databricks secrets put-secret <scope_name> access_key_id --string_value
"<your_aws_access_key_id>"
```

7. Store AWS secret access key using:

```
databricks secrets put-secret <scope_name> secret_access_key --string_value
"<your_aws_secret_access_key>"
```

In Databricks compute, set spark configuration as follows:

1. Log in to your Databricks workspace.
2. From the sidebar, select **Compute**.
3. In the list of available clusters, click the **target cluster's** name.
4. On the **Configuration** tab, expand **Advanced options**.
5. In the **Spark** tab, you can specify the spark configuration for the cluster.

To configure spark configuration for Databricks on Amazon S3:

```
fs.s3a.access.key {{secrets/<scope_name>/access_key_id}}
fs.s3a.secret.key {{secrets/<scope_name>/secret_access_key}}
fs.s3a.endpoint s3.amazonaws.com
fs.s3a.aws.credentials.provider
org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider
```

10.2.17.2.5.4 Creating Databricks Secrets

To create a Databricks secret:

- Log in to your Databricks workspace.
- In the sidebar, click **Compute**.
- In the list of available clusters, click the target cluster's name.
- On the **Apps** tab, click **Web Terminal**. This should open up a terminal session tab in the web browser.
- Create a Databricks secrets scope using `databricks secrets create-scope <scope-name>`. For example, `databricks secrets create-scope gg`.
- Create a secret using

```
databricks secrets put-secret --json '{
  "scope": "<scope-name>",
```

```
"key": "<key-name>",
"string_value": "<secret>"
}'
```

For example:

```
databricks secrets put-secret --json '{
  "scope": "gg",
  "key": "storageaccountname",
  "string_value": "-----storage-account-key-----"
}'
```

Note

These commands were run using Databricks Runtime (DBR) 15.0.

Note

On Unity Catalog enabled workspaces, the tables inside the hive_metastore catalog cannot use external location or external storage credentials. The tables inside the hive_metastore catalog also require the spark cluster configuration listed in this section.

10.2.17.2.6 Classpath Configuration

Databricks Event Handler uses the Databricks JDBC driver. Ensure that the classpath includes the path to the JDBC driver. You need to also include the respective object store Event Handler's dependencies in the classpath.

- [Dependencies](#)

10.2.17.2.6.1 Dependencies

Databricks JDBC driver: You can download the Dependency Downloader tool to download the JDBC driver by running the following script: `<OGGDIR>/DependencyDownloader/databricks.sh`.

Running this script without any input parameters will download the JDBC driver version 2.6.36. The script also can be run with a single argument to download a specific version of the JDBC driver.

For more information about Dependency Downloader, see [Dependency Downloader](#).

Alternatively, you can also download the JDBC driver from Maven central using the following co-ordinates:

```
<dependency>
  <groupId>com.databricks</groupId>
  <artifactId>databricks-jdbc</artifactId>
  <version>2.6.36</version>
</dependency>
```


- If staging location is set to ADLS Gen2, classpath should include the ABS Event handler dependencies. See [ABS Event handler](#) dependencies.
- If staging location is set to S3, classpath should include the S3 Event handler dependencies. See [S3 Event Handler](#).
- If staging location is set to GCS, classpath should include the GCS Event handler dependencies. See [GCS Event Handler](#).

Edit the `gg.classpath` configuration parameter to include the path to the object store Event Handler dependencies (if external stage is in use) and the Databricks JDBC driver.

10.2.17.2.7 Proxy Configuration

When the Replicat process runs behind a proxy server, the JDBC connection URL must be appended with the following property values:

- UseProxy
- ProxyHost
- ProxyPort

For example:

```
jdbc:databricks://<server-hostname>:443;httpPath=<http-  
path>[;<setting1>=<value1>;<setting2>=<value2>;<settingN>=<valueN>]  
;EnableArrow=0;UseProxy=1;ProxyHost=<proxy_host>;ProxyPort=<proxy_port>
```

10.2.17.2.8 INSERTALLRECORDS Support

Stage and merge targets supports INSERTALLRECORDS parameter.

See [INSERTALLRECORDS](#) in *Reference for Oracle GoldenGate*. Set the INSERTALLRECORDS parameter in the Replicat parameter file (.prm). Set the INSERTALLRECORDS parameter in the Replicat parameter file (.prm)

Setting this property directs the Replicat process to use bulk insert operations to load operation data into the target table. You can tune the batch size of bulk inserts using the File Writer property `gg.handler.databricks.maxFileSize`. The default value is set to 1GB. The frequency of bulk inserts can be tuned using the File writer property `gg.handler.databricks.fileRollInterval`, the default value is set to 3m (three minutes).

10.2.17.2.9 Operation Aggregation

Operation aggregation is the process of aggregating (merging/compressing) multiple operations on the same row into a single output operation based on a threshold.

- [In-memory Operation Aggregation](#)
- [Operation Aggregation using SQL](#)

10.2.17.2.9.1 In-memory Operation Aggregation

- Operation records can be aggregated in-memory, this is the default configuration.
- User can tune the frequency of merge interval using the File writer `gg.handler.databricks.fileRollInterval` property, the default value is set to 3m (three minutes).
- Operation aggregation in-memory requires additional JVM memory configuration.

10.2.17.2.9.2 Operation Aggregation using SQL

- To use SQL aggregation, it is mandatory that the trail files contain uncompressed UPDATE operation records, which means that the UPDATE operations contain full image of the row being updated.
- Operation aggregation using SQL can provide better throughput if the trails files contains uncompressed update records.
- Replicat can aggregate operations using SQL statements by setting the `gg.aggregate.operations.using.sql=true`.
- User can tune the frequency of merge interval using the File writer `gg.handler.databricks.fileRollInterval` property, the default value is set to 3m (three minutes).
- Operation aggregation using SQL does not require additional JVM memory configuration.

10.2.17.2.10 Compressed Update Handling

A compressed update record contains values for the key columns and the modified columns.

An uncompressed update record contains values for all the columns.

GoldenGate trails may contain compressed or uncompressed update records. The default extract configuration writes compressed updates to the trails.

The parameter `gg.compressed.update` can be set to true or false to indicate compressed/uncompressed update records.

- [MERGE Statement with Uncompressed Updates](#)

10.2.17.2.10.1 MERGE Statement with Uncompressed Updates

In some use cases, if the trail contains uncompressed update records, then the MERGE SQL statement can be optimized for better performance by setting `gg.compressed.update=false`

If you want to use DELETE+INSERT SQL statements instead of a MERGE SQL statement, then set `gg.eventhandler.databricks.deleteInsert=true`.

10.2.17.2.11 End-to-End Configuration

The following is an end-end configuration example which uses autoconfiguration.

The sample properties file can also be found in the directory `<OGGDIR>/AdapterExamples/big-data/databricks/`.

- **dbx-az.props**: Configuration using ADLS Gen2 stage for Databricks on Azure.
- **dbx-s3.props**: Configuration using S3 stage for Databricks on AWS.
- **dbx-gcs.props**: Configuration using GCS stage for Databricks on GCP.
- [Databricks on Azure](#)
- [Databricks on AWS](#)
- [Databricks on GCP](#)

10.2.17.2.11.1 Databricks on Azure

```
# Configuration to load GoldenGate trail operation records into Databricks using ADLS
Gen2 stage.
# Note: Recommended to only edit the configuration marked as  TODO
gg.target=databricks
# Azure Blob Event handler.
gg.eventhandler.abs.bucketMappingTemplate=<azure_adls_gen2_container_name>
gg.eventhandler.abs.accountName=<azure_storage_account_name>
gg.eventhandler.abs.accountKey=<azure_storage_account_key>

# Databricks Event Handler.
#TODO: Edit JDBC ConnectionUrl
gg.eventhandler.databricks.connectionURL=jdbc:databricks://<server-
hostname>:443;httpPath=<http-
path>[;<setting1>=<value1>;<setting2>=<value2>;<settingN>=<valueN>];EnableArrow=0
#TODO: Edit JDBC username or 'token'
gg.eventhandler.databricks.UserName=token
#TODO: Edit JDBC password
gg.eventhandler.databricks.Password=<password>

#TODO: Edit the classpath to include Azure Blob Event Handler dependencies and
Databricks JDBC driver.
gg.classpath=$THIRD_PARTY_DIR/abs/*:$THIRD_PARTY_DIR/databricks/*
#TODO: Provide sufficient memory (at least 8GB).
jvm.bootoptions=-Xmx8g -Xms1g
```

10.2.17.2.11.2 Databricks on AWS

```
# Configuration to load GoldenGate trail operation records into Databricks using S3
stage.
# Note: Recommended to only edit the configuration marked as  TODO
gg.target=databricks
gg.stage=s3
#The S3 Event Handler
#TODO: Edit the AWS region
gg.eventhandler.s3.region=<aws region>
#TODO: Edit the AWS S3 bucket
gg.eventhandler.s3.bucketMappingTemplate=<s3 bucket>

# Databricks Event Handler.
#TODO: Edit JDBC ConnectionUrl
gg.eventhandler.databricks.connectionURL=jdbc:databricks://<server-
hostname>:443;httpPath=<http-
path>[;<setting1>=<value1>;<setting2>=<value2>;<settingN>=<valueN>];EnableArrow=0
#TODO: Edit JDBC username or 'token'
gg.eventhandler.databricks.UserName=token
#TODO: Edit JDBC password
gg.eventhandler.databricks.Password=<password>

#TODO: Edit the classpath to include GCS Event Handler dependencies and Databricks JDBC
driver.
gg.classpath=$THIRD_PARTY_DIR/s3/*:$THIRD_PARTY_DIR/databricks/*
#TODO: Provide sufficient memory (at least 8GB).
jvm.bootoptions=-Xmx8g -Xms1g
```

10.2.17.2.11.3 Databricks on GCP

```
# Configuration to load GoldenGate trail operation records into Databricks using GCS
stage.
# Note: Recommended to only edit the configuration marked as  TODO
```

```

gg.target=databricks
gg.stage=gcs
## The GCS Event handler
#TODO: Edit the GCS bucket name
gg.eventhandler.gcs.bucketMappingTemplate=<gcs bucket>
#TODO: Edit the GCS credentialsFile
gg.eventhandler.gcs.credentialsFile=<oggbd-project-credentials.json>

# Databricks Event Handler.
#TODO: Edit JDBC ConnectionUrl
gg.eventhandler.databricks.connectionURL=jdbc:databricks://<server-
hostname>:443;httpPath=<http-
path>[;<setting1>=<value1>;<setting2>=<value2>;<settingN>=<valueN>];EnableArrow=0
#TODO: Edit JDBC username or 'token'
gg.eventhandler.databricks.UserName=token
#TODO: Edit JDBC password
gg.eventhandler.databricks.Password=<password>

#TODO: Edit the classpath to include GCS Event Handler dependencies and Databricks JDBC
driver.
gg.classpath=$THIRD_PARTY_DIR/gcs/*:$THIRD_PARTY_DIR/databricks/*
#TODO: Provide sufficient memory (at least 8GB).
jvm.bootoptions=-Xmx8g -Xms1g

```

10.2.17.2.12 Table Mapping

If the MAP statement does not specify a target catalog, then the default catalog for the Databricks workspace will be used. The handler will log the default catalog during initialization. Example log message: Connection catalog is set to [dbx-catalog].

- [Mapping Table](#)

10.2.17.2.12.1 Mapping Table

Table 10-16

MAP statement in the Replicat parameter file	Databricks Catalog	Databricks Schema	Databricks Table
MAP SCHEMA_1.TABLE_1, TARGET "schema_1"."table_1" ";	Default catalog	schema_1	table_1
MAP DB_1.SCHEMA_1.TABLE_1, TARGET "db_1"."schema_1"."table_1"	db_1	schema_1	table_1

10.2.17.3 Troubleshooting and Diagnostics

- **Unsupported Databricks data types:**
 - ARRAY
 - MAP
 - STRUCT

- INTERVAL
- VOID
- **Databricks JDBC Driver Exception:**

```
java.lang.ClassCastException: class
    org.apache.logging.log4j.core.lookup.ResourceBundleLookup
```

While using the Databricks JDBC driver 2.6.36, we have come across this exception. Setting the property `EnableArrow=0` is the workaround.

Oracle is working with Databricks to address this.

- **org.apache.hive.service.cli.HiveSQLException: Error running query:**
org.apache.spark.sql.AnalysisException: Cannot add column '<column name>' with type 'void'. Please explicitly specify a non-void type.

If the Databricks table defined a column with data type VOID, all the DML operations will fail even if the VOID column is not mapped by Oracle GoldenGate.

To proceed with the replication, you should drop the VOID column from the target table.

- **Databricks INTERVAL data type:**
If the target table contains an INTERVAL type, the Databricks JDBC driver ignores the presence of such a column.

You cannot map any source column to an INTERVAL type. You should also avoid defining INTERVAL types in the target table without a default value and with a NOT NULL constraint.
- **Connectivity issues to Databricks:**
 - Validate JDBC connection URL, username, and password.
 - Check proxy configuration if running Replicat process behind a proxy.
- **DDL not applied on the target table:** GoldenGate for Distributed Applications and Analytics does not support DDL replication.
- **SQL Errors:** In case there are any errors while executing any SQL, the SQL statements along with the bind parameter values are logged into the GoldenGate for Distributed Applications and Analytics handler log file.
- **Co-existence of the components:** When using an external stage location (ADLS Gen 2 or S3 or GCS), the location/region of the machine where Replicat process is running and the object store's region will have an impact on the overall throughput of the apply process. For the best possible throughput, the components need to be located ideally in the same region or as close as possible.
- **Replicat ABEND due to partial LOB records in the trail file:** Oracle GoldenGate for Distributed Applications and Analytics does not support replication of partial LOB data. The trail file needs to be regenerated by Oracle Integrated capture using `TRANLOGOPTIONS FETCHPARTIALLOB` option in the Extract parameter file.
- When replicating to more than ten target tables, the parameter `maxConnections` can be increased to a higher value which can improve throughput.
- **Identity column in the target table:**
 - If the target table contains an Identity column, then the MERGE statement would fail because the Identity cannot be updated.

Example Error Message:

```
`Query: MERGE INTO***, Error message
      from Server: org.apache.hive.service.cli.HiveSQLException: Error
running query:
      org.apache.spark.sql.AnalysisException:    UPDATE on IDENTITY
column "col9" is not
      supported.`
```

To proceed, review the following points:

- If the Identity column is defined using `GENERATED ALWAYS AS IDENTITY`, then Replicat would result in the following error: **Error message from Server:**
`org.apache.hive.service.cli.HiveSQLException: Error running query:`
`org.apache.spark.sql.AnalysisException: Providing values for GENERATED`
`ALWAYS AS IDENTITY column col9 is not supported. To proceed further, the Identity`
`column should be excluded from mapping on the source database using COLSEXCEPT`
`or removed from the target table.`
- If the Identity column is defined using `GENERATED BY DEFAULT AS IDENTITY`, then Replicat can be configured to use `DELETE-INSERT` instead of `MERGE` by setting `gg.eventhandler.databricks.deleteInsert=true` provided the prerequisites for enabling `DELETE-INSERT` are met.
- **org.apache.spark.SparkException: [FAILED_READ_FILE.FILE_NOT_EXIST] Error while reading file:** In case of a high load on the cluster, it is observed that calls to refresh the contents of the external table are not reliable and lead to the preceding error and a replicatabend due to Databricks limitations. The alternate solution is to configure `gg.eventhandler.databricks.externalTableRefreshMethod=ALTER`.

10.2.17.4 Restrictions on Database Object Names

- **Catalog, Schema, and Table Names:** Cannot contain space, period (`.`), slash (`/`), asterisk (`*`), double quote (`"`), backtick (```), question mark (`?`), backslash (`\`), colon (`:`), dollar (`$`), percent (`%`), ampersand (`&`), hash (`#`), at (`@`), exclamation (`!`), tilde (`~`), open parenthesis (`(`), close parenthesis (`)`), open bracket (`[`), close bracket (`]`), open brace (`{`), close brace (`}`), pipe (`|`), semicolon (`;`), less than (`<`), greater than (`>`), comma (`,`), or control characters (ASCII 0-31).
- **Table Names:** Cannot contain space, period (`.`), asterisk (`*`), question mark (`?`), double quote (`"`), backslash (`\`), backtick (```), slash (`/`), colon (`:`), or control characters (ASCII 0-31).
- **Specific limitation for GGDAA 21c and previous releases:** Supports only names that comply with the Avro specification for identifier names: must start with `[A-Za-z_]``, followed by `[A-Za-z0-9_]``
- Additionally, all object names are also limited by the supported list of characters for the respective target.

10.2.18 DataStax

Datastax Enterprise is a NoSQL database built on Apache Cassandra. For more information, see [Apache Cassandra](#) for configuring replication to Datastax Enterprise.

10.2.19 Confluent Kafka

- Confluent is a primary adopter of Kafka Connect and their Confluent Platform offering includes extensions over the standard Kafka Connect functionality. This includes Avro serialization and deserialization, and an Avro schema registry. Much of the Kafka Connect functionality is available in Apache Kafka.
- You can use Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) [Kafka Connect Handler](#) to replicate to Confluent Kafka. The [Kafka Connect Handler](#) is a Kafka Connect source connector. You can capture database changes from any database supported by Oracle GoldenGate and stream that change of data through the Kafka Connect layer to Kafka.
- Kafka Connect uses proprietary objects to define the schemas (`org.apache.kafka.connect.data.Schema`) and the messages (`org.apache.kafka.connect.data.Struct`). The [Kafka Connect Handler](#) can be configured to manage what data is published and the structure of the published data.
- The [Kafka Connect Handler](#) does not support any of the pluggable formatters that are supported by the Kafka Handler.

10.2.20 Elasticsearch

- [Elasticsearch with Elasticsearch 7x and 6x](#)
The Elasticsearch Handler allows you to store, search, and analyze large volumes of data quickly and in near real time.
- [Elasticsearch 8x](#)
The Elasticsearch Handler allows you to store, search, and analyze large volumes of data quickly and in near real time.
- [Support for Vector Data](#)

10.2.20.1 Elasticsearch with Elasticsearch 7x and 6x

The Elasticsearch Handler allows you to store, search, and analyze large volumes of data quickly and in near real time.

This article describes how to use the Elasticsearch handler.

Note

This section on the Elasticsearch Handler pertains to Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) versions 21.9.0.0.0 and before. Starting with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 21.10.0.0.0, the Elasticsearch client was changed in order to support Elasticsearch 8.x.

- [Overview](#)
- [Detailing the Functionality](#)
- [Setting Up and Running the Elasticsearch Handler](#)
- [Troubleshooting](#)
- [Performance Consideration](#)

- [About the Shield Plug-In Support](#)
- [About DDL Handling](#)
- [Known Issues in the Elasticsearch Handler](#)
- [Elasticsearch Handler Transport Client Dependencies](#)
What are the dependencies for the Elasticsearch Handler to connect to Elasticsearch databases?
- [Elasticsearch High Level REST Client Dependencies](#)

10.2.20.1.1 Overview

Elasticsearch is a highly scalable open-source full-text search and analytics engine. Elasticsearch allows you to store, search, and analyze large volumes of data quickly and in near real time. It is generally used as the underlying engine or technology that drives applications with complex search features.

The Elasticsearch Handler uses the Elasticsearch Java client to connect and receive data into Elasticsearch node, see <https://www.elastic.co>.

Note

This section on the Elasticsearch Handler pertains to Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) versions 21.9.0.0.0 and before. Starting with GG for DAA 21.10.0.0.0, the Elasticsearch client was changed in order to support Elasticsearch 8.x.

10.2.20.1.2 Detailing the Functionality

This topic details the Elasticsearch Handler functionality.

- [About the Elasticsearch Version Property](#)
- [About the Index and Type](#)
- [About the Document](#)
- [About the Primary Key Update](#)
- [About the Data Types](#)
- [Operation Mode](#)
- [Operation Processing Support](#)
- [About the Connection](#)

10.2.20.1.2.1 About the Elasticsearch Version Property

The Elasticsearch Handler supports two different clients to communicate with the Elasticsearch cluster: The Elasticsearch transport client and the Elasticsearch High Level REST client.

Elasticsearch Handler can also be configured for the two supported clients by specifying the appropriate version of Elasticsearch handler properties file. Older version of Elasticsearch (6.x) supports only Transport client and the Elasticsearch handler can be configured by setting the configurable property version value to 6.x. For the latest version of Elasticsearch (7.x), both the Transport client and the High Level REST client are supported. Therefore, in the latest version, the Elasticsearch Handler can be configured for Transport client by setting the value of

configurable property version to 7.x and High Level REST client by setting the value to Rest7.x.

The configurable parameters for each of them are as follows:

1. Set the `gg.handler.name.version` configuration value to 6.x or 7.x to connect to the Elasticsearch cluster using the transport client using the respective version.
2. Set the `gg.handler.name.version` configuration value to REST7.0 to connect to the Elasticsearch cluster using the Elasticsearch High Level REST client. The REST client support Elasticsearch versions 7.x.

10.2.20.1.2.2 About the Index and Type

An Elasticsearch **index** is a collection of documents with similar characteristics. An index can only be created in lowercase. An Elasticsearch **type** is a logical group within an index. All the documents within an index or type should have same number and type of fields.

The Elasticsearch Handler maps the source trail schema concatenated with source trail table name to construct the index. For three-part table names in source trail, the index is constructed by concatenating source catalog, schema, and table name.

The Elasticsearch Handler maps the source table name to the Elasticsearch type. The type name is case-sensitive.

Note

Elasticsearch field names are case sensitive. If the field name in the data to be either updated or inserted are in uppercase and the existing fields in Elasticsearch server are in lowercase, then they are treated as new fields and not updated as existing fields. The workaround for this is using the parameter `gg.schema.normalize=lowercase`, which will update the field name to lowercase, thus resolving the issue.

Table 10-17 Elasticsearch Mapping

Source Trail	Elasticsearch Index	Elasticsearch Type
<code>schema.tablename</code>	<code>schema_tablename</code>	<code>tablename</code>
<code>catalog.schema.tablename</code>	<code>catalog_schema_tablename</code>	<code>tablename</code>

If an index does not already exist in the Elasticsearch cluster, a new index is created when Elasticsearch Handler receives (INSERT or UPDATE operation in source trail) data.

10.2.20.1.2.3 About the Document

An Elasticsearch document is a basic unit of information that can be indexed. Within an index or type, you can store as many documents as you want. Each document has a unique identifier based on the `_id` field.

The Elasticsearch Handler maps the source trail primary key column value as the document identifier.

10.2.20.1.2.4 About the Primary Key Update

The Elasticsearch document identifier is created based on the source table's primary key column value. The document identifier cannot be modified. The Elasticsearch handler

processes a source primary key's update operation by performing a DELETE followed by an INSERT. While performing the INSERT, there is a possibility that the new document may contain fewer fields than required. For the INSERT operation to contain all the fields in the source table, enable trail Extract to capture the full data before images for update operations or use GETBEFORECOLS to write the required column's before images.

10.2.20.1.2.5 About the Data Types

Elasticsearch supports the following data types:

- 32-bit integer
- 64-bit integer
- Double
- Date
- String
- Binary

10.2.20.1.2.6 Operation Mode

The Elasticsearch Handler uses the operation mode for better performance. The `gg.handler.name.mode` property is not used by the handler.

10.2.20.1.2.7 Operation Processing Support

The Elasticsearch Handler maps the source table name to the Elasticsearch type. The type name is case-sensitive.

For three-part table names in source trail, the index is constructed by concatenating source catalog, schema, and table name.

INSERT

The Elasticsearch Handler creates a new index if the index does not exist, and then inserts a new document.

UPDATE

If an Elasticsearch index or document exists, the document is updated. If an Elasticsearch index or document does not exist, a new index is created and the column values in the UPDATE operation are inserted as a new document.

DELETE

If an Elasticsearch index or document exists, the document is deleted. If Elasticsearch index or document does not exist, a new index is created with zero fields.

The TRUNCATE operation is not supported.

10.2.20.1.2.8 About the Connection

A **cluster** is a collection of one or more nodes (servers) that holds the entire data. It provides federated indexing and search capabilities across all nodes.

A **node** is a single server that is part of the cluster, stores the data, and participates in the cluster's indexing and searching.

The Elasticsearch Handler property `gg.handler.name.ServerAddressList` can be set to point to the nodes available in the cluster.

10.2.20.1.3 Setting Up and Running the Elasticsearch Handler

You must ensure that the Elasticsearch cluster is setup correctly and the cluster is up and running, see <https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>. Alternatively, you can use Kibana to verify the setup.

Set the Classpath

The property `gg.classpath` must include all the jars required by the Java transport client. For a listing of the required client JAR files by version, see [Elasticsearch Handler Transport Client Dependencies](#). For a listing of the required client JAR files for the Elasticsearch High Level REST client, see [Elasticsearch High Level REST Client Dependencies](#).

The inclusion of the `*` wildcard in the path can include the `*` wildcard character in order to include all of the JAR files in that directory in the associated classpath. Do not use `*.jar`.

The following is an example of the correctly configured classpath:

```
gg.classpath=Elasticsearch_Home/lib/*
```

- [Configuring the Elasticsearch Handler](#)

10.2.20.1.3.1 Configuring the Elasticsearch Handler

Elasticsearch Handler can be configured for different version of Elasticsearch. For the latest version (7.x), two types of clients are supported: the Transport client and High-level REST client. When the configurable property `version` is set to the values 6.x or 7.x it uses Elasticsearch Transport client for connecting and performing all other operations of handler to Elasticsearch cluster. When the configurable property `version` is set to `rest7.x`, it uses Elasticsearch High Level REST client for connecting and performing other operations of handler to Elasticsearch 7.x cluster. The configurable parameters for each of them are separately given below:

Table 10-18 Common Configurable Properties

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.handlerlist</code>	Required	Name (Any name of your choice for handler)	None	The list of handlers to be used.
<code>gg.handler.<name>.type</code>	Required	elasticsearch	None	Type of handler to use. For example, Elasticsearch, Kafka, or Flume.
<code>gg.handler.name.ServerAddressList</code>	Optional	<i>Server:Port</i> [, <i>Server:Port ...</i>]	• localhost:9300 (for Transport Client) • localhost:9200 (for High-Level REST Client)	Comma separated list of contact points of the nodes. The allowed port for version REST7.x is 9200. For other version, it is 9300.

Table 10-18 (Cont.) Common Configurable Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.version</code>	Required	5.x 6.x 7.x REST7.x	7.x	The version values 5.x, 6.x, and 7.x indicate using the Elasticsearch Transport client to communicate with Elasticsearch version 5.x, 6.x and 7.x respectively. The version REST7.x indicates using the Elasticsearch High Level REST client to communicate with Elasticsearch version 7.x.
<code>gg.handler.name</code> <code>.version</code> <code>gg.handler.name</code> <code>.bulkWrite</code>	Optional	true false	false	When this property is true, the Elasticsearch Handler uses the bulk write API to ingest data into Elasticsearch cluster. The batch size of bulk write can be controlled using the MAXTRANSOPS Replicat parameter.
<code>gg.handler.name</code> <code>.numberAsString</code>	Optional	true false	false	When this property is true, the Elasticsearch Handler receives all the number column values (Long, Integer, or Double) in the source trail as strings into the Elasticsearch cluster.
<code>gg.handler.elas</code> <code>ticsearch.upser</code> <code>t</code>	Optional	true false	true	When this property is true, a new document is inserted if the document does not already exist when performing an UPDATE operation.

Example 10-1 Sample Handler Properties file:

Sample Replicat configuration and a Java Adapter Properties files can be found at the following directory:

GoldenGate_install_directory/AdapterExamples/big-data/elasticsearch
For Elasticsearch REST handler

```
gg.handlerlist=elasticsearch
gg.handler.elasticsearch.type=elasticsearch
gg.handler.elasticsearch.ServerAddressList=localhost:9300
gg.handler.elasticsearch.version=rest7.x
gg.classpath=/path/to/elasticsearch/lib/*:/path/to/elasticsearch/modules/reindex/*:/
path/to/elasticsearch/modules/lang-mustache/*:/path/to/elasticsearch/modules/rank-eval/*
```

- [Common Configurable Properties](#)
- [Transport Client Configurable Properties](#)
- [Transport Client Setting Properties File](#)
- [Classpath Settings for Transport Client](#)
- [REST Client Configurable Properties](#)
- [Authentication for REST Client](#)
- [Classpath Settings for REST Client](#)

10.2.20.1.3.1.1 Common Configurable Properties

The common configurable properties that are applicable for all the versions of Elasticsearch and applicable for both Transport client as well as High Level REST client of Elasticsearch handler are as shown in the following table:

Table 10-19 Common Configurable Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handlerlist</code>	Required	Name (Any name of your choice for handler)	None	The list of handlers to be used.
<code>gg.handler.<name>.type</code>	Required	elasticsearch	None	Type of handler to use. For example, Elasticsearch, Kafka, or Flume.
<code>gg.handler.name.ServerAddressList</code>	Optional	<i>Server:Port</i> [, <i>Server:Port ...</i>]	- localhost:9300 (for Transport Client) - localhost:9200 (for High-Level REST Client)	Comma separated list of contact points of the nodes. The allowed port for version REST7.x is 9200. For other version, it is 9300.

Table 10-19 (Cont.) Common Configurable Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.version</code>	Required	6.x 7.x REST7.x	7.x	The version values 6.x, and 7.x indicate using the Elasticsearch Transport client to communicate with Elasticsearch version 6.x and 7.x respectively. The version REST7.x indicates using the Elasticsearch High Level REST client to communicate with Elasticsearch version 7.x.
<code>gg.handler.name</code> <code>.version</code> <code>gg.handler.name</code> <code>.bulkWrite</code>	Optional	true false	false	When this property is true, the Elasticsearch Handler uses the bulk write API to ingest data into Elasticsearch cluster. The batch size of bulk write can be controlled using the MAXTRANSOPS Replicat parameter.
<code>gg.handler.name</code> <code>.numberAsString</code>	Optional	true false	false	When this property is true, the Elasticsearch Handler receives all the number column values (Long, Integer, or Double) in the source trail as strings into the Elasticsearch cluster.
<code>gg.handler.elasticsearch.upsert</code>	Optional	true false	true	When this property is true, a new document is inserted if the document does not already exist when performing an UPDATE operation.

10.2.20.1.3.1.2 Transport Client Configurable Properties

When the configurable property version is set to the value 6.x or 7.x, it uses Transport client to communicate with the corresponding version of Elasticsearch cluster. The configurable properties applicable when using Transport client only are as follows:

Table 10-20 Transport Client Configurable Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.clientSettingsFile</code>	Required	Transport client properties file.	None	The filename in classpath that holds Elasticsearch transport client properties used by the Elasticsearch Handler.

```
Copygg.handlerlist=elasticsearch
gg.handler.elasticsearch.type=elasticsearch
gg.handler.elasticsearch.ServerAddressList=localhost:9300
gg.handler.elasticsearch.clientSettingsFile=client.properties
gg.handler.elasticsearch.version=[6.x | 7.x]
gg.classpath=/path/to/elastic/lib/*:/path/to/elastic/modules/transport-netty4/*:/path/to/elastic/modules/reindex/*:/path/to/elastic/plugins/x-pack/*:
```

10.2.20.1.3.1.3 Transport Client Setting Properties File

The Elasticsearch Handler uses a Java Transport client to interact with Elasticsearch cluster. The Elasticsearch cluster may have additional plug-ins like shield or x-pack, which may require additional configuration.

The `gg.handler.name.clientSettingsFile` property should point to a file that has additional client settings based on the version of Elasticsearch cluster.

The Elasticsearch Handler attempts to locate and load the client settings file using the Java classpath. The Java classpath must include the directory containing the properties file. The client properties file for Elasticsearch (without any plug-in) is:

```
cluster.name=Elasticsearch_cluster_name.
```

The Shield plug-in also supports additional capabilities like SSL and IP filtering. The properties can be set in the `client.properties` file, see https://www.elastic.co/guide/en/shield/current/using_elasticsearch_java_clients_with_shield.html.

Example of `client.properties` file for Elasticsearch Handler with X-Pack plug-in:

```
Copycluster.name=Elasticsearch_cluster_name
xpack.security.user=x-pack_username:x-pack-password
```

The X-Pack plug-in also supports additional capabilities. The properties can be set in the `client.properties` file, see

<https://www.elastic.co/guide/en/elasticsearch/client/java-api/5.1/transport-client.html> and <https://www.elastic.co/guide/en/x-pack/current/java-clients.html>

10.2.20.1.3.1.4 Classpath Settings for Transport Client

The `gg.classpath` setting for Elasticsearch handler with Transport client should contain the path to jars from library (lib) and modules (transport-netty4 and reindex modules) folder inside Elasticsearch installation directory. If x-pack plugin is used for authentication purpose, then the classpath should also include the jars inside the plugins (x-pack) folder inside Elasticsearch installation directory. See the path for jars as follows:

.

1. [path/to/elastic/lib/*]
2. [/path/to/elastic/modules/transport-netty4/*]
3. [/path/to/elastic/modules/reindex/*]
4. [/path/to/elastic/plugins/x-pack/*] This needs to be added only if x-pack plugin is configured in Elasticsearch

10.2.20.1.3.1.5 REST Client Configurable Properties

When the configurable property version is set to value rest7.x, the handler uses Elasticsearch High Level REST client to connect to Elasticsearch 7.x cluster. The configurable properties that are supported for REST client only are as follows:

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler.elasticsearch.routingTemplate	Optional	\$ {columnValue[table1=column1, table2=column2, ...]}	None	The template to be used for deciding the routing algorithm.
gg.handler.name.authType	Optional	none basic ssl	None	Controls the authentication type for the Elasticsearch REST client. • none - No authentication • basic - Client authentication using username and password without message encryption. • ssl - Mutual authentication. Client authenticates the server using a trust-store. Server authentication client using username and password. Messages are encrypted.
gg.handler.name.authType	Required (for auth-type basic.)	A valid username	None	The username for the server to authenticate the Elasticsearch REST client. Must be provided for auth types basic.
gg.handler.name.basicAuthUsername				

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.basicAuthPassword</code>	Required (for auth-type basic.)	A valid password	None	The password for the server to authenticate the Elasticsearch REST client. Must be provided for auth types basic.
<code>gg.handler.name.trustStore</code>	Required (for auth-type SSL)	The fully qualified name (path + name) of trust-store file	None	The truststore for the Elasticsearch client to validate the certificate received from the Elasticsearch server. Must be provided if the auth type is set to ssl. Valid only for the Elasticsearch REST client
<code>gg.handler.name.trustStorePassword</code>	Required (for auth-type SSL)	A valid trust-store Password	None	The password for the truststore for the Elasticsearch REST client to validate the certificate received from the Elasticsearch server. Must be provided if the auth type is set to ssl.
<code>gg.handler.name.maxConnectTimeout</code>	Optional	Positive integer	Default value of Apache HTTP Components framework.	Set the maximum wait period for a connection to be established from the Elasticsearch REST client to the Elasticsearch server. Valid only for the Elasticsearch REST client.
<code>gg.handler.name.maxSocketTimeout</code>	Optional	Positive Integer	Default value of Apache HTTP Components framework.	Sets the maximum wait period in milliseconds to wait for a response from the service after issuing a request. May need to be increased when pushing large data volumes. Valid only for the Elasticsearch REST client.

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.proxyUsername</code>	Optional	The proxy server username	None	If the connectivity to the Elasticsearch uses the REST client and routing through a proxy server, then this property sets the username of your proxy server. Most proxy servers do not require credentials.
<code>gg.handler.name.proxyPassword</code>	Optional	The proxy server password	None	If the connectivity to the Elasticsearch uses the REST client and routing through a proxy server, then this property sets the password of your proxy server. Most proxy servers do not require credentials.
<code>gg.handler.name.proxyProtocol</code>	Optional	<code>http https</code>	None	If the connectivity to the Elasticsearch uses the REST client and routing through a proxy server, then this property sets the protocol of your proxy server.
<code>gg.handler.name.proxyPort</code>	Optional	The port number of your proxy server.	None	If the connectivity to the Elasticsearch uses the REST client and routing through a proxy server, then this property sets the port number of your proxy server.
<code>gg.handler.name.proxyServer</code>	Optional	The host name of your proxy server.	None	If the connectivity to the Elasticsearch uses the REST client and routing through a proxy server, then this property sets the host name of your proxy server.

Sample Properties for Elasticsearch Handler using REST Client

```
gg.handlerlist=elasticsearch
gg.handler.elasticsearch.type=elasticsearch
gg.handler.elasticsearch.ServerAddressList=localhost:9200
```

```
gg.handler.elasticsearch.version=rest7.x
gg.classpath=/path/to/elasticsearch/lib/*:/path/to/elasticsearch/modules/reindex/*:/
path/to/elasticsearch/modules/lang-mustache/*:/path/to/elasticsearch/modules/rank-eval/*
```

10.2.20.1.3.1.6 Authentication for REST Client

The configurable property `auth-type` value `SSL` can be used to configure the SSL authentication mechanism for communicating with Elasticsearch cluster. This property can also be used to configure the basic authentication with SSL by providing configurable property `basic-username/password` along with the `trust-store` properties.

10.2.20.1.3.1.7 Classpath Settings for REST Client

The Classpath for High Level REST client must contain the jars from the library (`lib`) folder and modules folders (`reindex`, `lang-mustache` and `ran-eval`) inside the Elasticsearch installation directory. The REST client are dependent on these libraries and should be included in `gg.classpath` for the handler to work. Following are the list of dependencies:

1. `[/path/to/elasticsearch/lib/*]`
2. `[/path/to/elasticsearch/modules/reindex/*]`
3. `[/path/to/elasticsearch/modules/lang-mustache/*]`
4. `[/path/to/elasticsearch/modules/rank-eval/*]`

10.2.20.1.4 Troubleshooting

This section contains information to help you troubleshoot various issues.

Transport Client Properties File Not Found

This is applicable for Transport Client only when the property `version` is set to 6.x or 7.x.

Error:

```
ERROR 2017-01-30 22:33:10,058 [main] Unable to establish connection. Check handler
properties
and client settings configuration.
```

To resolve this exception, verify that the `gg.handler.name.clientSettingsFile` configuration property is correctly setting the Elasticsearch transport client settings file name. Verify that the `gg.classpath` variable includes the path to the correct file name and that the path to the properties file does not contain an asterisk (*) wildcard at the end.

- [Incorrect Java Classpath](#)
- [Elasticsearch Version Mismatch](#)
- [Transport Client Properties File Not Found](#)
- [Cluster Connection Problem](#)
- [Unsupported Truncate Operation](#)
- [Bulk Execute Errors](#)

10.2.20.1.4.1 Incorrect Java Classpath

The most common initial error is an incorrect classpath to include all the required client libraries and creates a `ClassNotFoundException` in the `log4j` log file.

Also, it may be due to an error resolving the classpath if there is a typographic error in the `gg.classpath` variable.

The Elasticsearch transport client libraries do not ship with the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) product. You should properly configure the

`gg.classpath` property in the Java Adapter Properties file to correctly resolve the client libraries, see [Setting Up and Running the Elasticsearch Handler](#).

10.2.20.1.4.2 Elasticsearch Version Mismatch

The Elasticsearch Handler `gg.handler.name.version` property must be set to one of the following values: 6.x, 7.x or REST7.x to match the major version number of the Elasticsearch cluster. For example, `gg.handler.name.version=7.x`.

The following errors may occur when there is a wrong version configuration:

```
Error: NoNodeAvailableException[None of the configured nodes are available:]
```

```
ERROR 2017-01-30 22:35:07,240 [main] Unable to establish connection. Check handler properties and client settings configuration.
```

```
java.lang.IllegalArgumentException: unknown setting [shield.user]
```

Ensure that all required plug-ins are installed and review documentation changes for any removed settings.

10.2.20.1.4.3 Transport Client Properties File Not Found

To resolve this exception:

```
ERROR 2017-01-30 22:33:10,058 [main] Unable to establish connection. Check handler properties and client settings configuration.
```

Verify that the `gg.handler.name.clientSettingsFile` configuration property is correctly setting the Elasticsearch transport client settings file name. Verify that the `gg.classpath` variable includes the path to the correct file name and that the path to the properties file does not contain an asterisk (*) wildcard at the end.

10.2.20.1.4.4 Cluster Connection Problem

This error occurs when the Elasticsearch Handler is unable to connect to the Elasticsearch cluster:

```
Error: NoNodeAvailableException[None of the configured nodes are available:]
```

Use the following steps to debug the issue:

1. Ensure that the Elasticsearch server process is running.
2. Validate the `cluster.name` property in the client properties configuration file.
3. Validate the authentication credentials for the x-Pack or Shield plug-in in the client properties file.
4. Validate the `gg.handler.name.ServerAddressList` handler property.

10.2.20.1.4.5 Unsupported Truncate Operation

The following error occurs when the Elasticsearch Handler finds a TRUNCATE operation in the source trail:

```
oracle.goldengate.util.GGException: Elasticsearch Handler does not support the operation: TRUNCATE
```

This exception error message is written to the handler log file before the RAeplicat process abends. Removing the GETTRUNCATES parameter from the Replicat parameter file resolves this error.

10.2.20.1.4.6 Bulk Execute Errors

....

```
DEBUG [main] (ElasticSearch5DOTX.java:130) - Bulk execute status: failures:
[true] buildFailureMessage:[failure in bulk execution: [0]: index
[cs2cat_s1sch_n1tab], type [N1TAB], id [83], message
[RemoteTransportException[[U0vac8l][127.0.0.1:9300][indices:data/write/bulk[s]
[p]]]]; nested: EsRejectedExecutionException[rejected execution of
org.elasticsearch.transport.TransportService$7@43eddfb2 on
EsThreadPoolExecutor[bulk, queue capacity = 50,
org.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@5ef5f412[Running
, pool size = 4, active threads = 4, queued tasks = 50, completed tasks =
84]]];]
```

It may be due to the Elasticsearch running out of resources to process the operation. You can limit the Replicat batch size using MAXTRANSOPS to match the value of the `thread_pool.bulk.queue_size` Elasticsearch configuration parameter.

Note

Changes to the Elasticsearch parameter, `thread_pool.bulk.queue_size`, are effective only after the Elasticsearch node is restarted.

10.2.20.1.5 Performance Consideration

The Elasticsearch Handler `gg.handler.name.bulkWrite` property is used to determine whether the source trail records should be pushed to the Elasticsearch cluster one at a time or in bulk using the bulk write API. When this property is **true**, the source trail operations are pushed to the Elasticsearch cluster in batches whose size can be controlled by the MAXTRANSOPS parameter in the generic Replicat parameter file. Using the bulk write API provides better performance.

Elasticsearch uses different thread pools to improve how memory consumption of threads are managed within a node. Many of these pools also have queues associated with them, which allow pending requests to be held instead of discarded.

For bulk operations, the default queue size is 50 (in version 5.2) and 200 (in version 5.3).

To avoid bulk API errors, you must set the Replicat MAXTRANSOPS size to match the bulk thread pool queue size at a minimum. The configuration `thread_pool.bulk.queue_size` property can be modified in the `elasticsearch.yaml` file.

10.2.20.1.6 About the Shield Plug-In Support

Elasticsearch versions 6.x and 7.x (X-Pack plug-in for Elasticsearch 6.x and 7.x) support a Shield plug-in which provides basic authentication, SSL and IP filtering. Similar capabilities exist in the X-Pack plug-in for Elasticsearch 6.x and 7.x. The additional transport client settings can be configured in the Elasticsearch Handler using the `gg.handler.name.clientSettingsFile` property.

10.2.20.1.7 About DDL Handling

The Elasticsearch Handler does not react to any DDL records in the source trail. Any data manipulation records for a new source table results in auto-creation of index or type in the Elasticsearch cluster.

10.2.20.1.8 Known Issues in the Elasticsearch Handler

Elasticsearch: Trying to input very large number

Very large numbers result in inaccurate values with Elasticsearch document. For example, 9223372036854775807, -9223372036854775808. This is an issue with the Elasticsearch server and not a limitation of the Elasticsearch Handler.

The workaround for this issue is to ingest all the number values as strings using the `gg.handler.name.numberAsString=true` property.

Elasticsearch: Issue with index

The Elasticsearch Handler is not able to input data into the same index if there are more than one table with similar column names and different column data types.

Index names are always lowercase though the `catalog/schema/tablename` in the trail may be case-sensitive.

10.2.20.1.9 Elasticsearch Handler Transport Client Dependencies

What are the dependencies for the Elasticsearch Handler to connect to Elasticsearch databases?

The maven central repository artifacts for Elasticsearch databases are:

Maven groupId: `org.elasticsearch.client`

Maven artifactId: `transport`

Maven groupId: `org.elasticsearch.client`

Maven artifactId: `x-pack-transport`

10.2.20.1.10 Elasticsearch High Level REST Client Dependencies

The maven coordinates for the Elasticsearch High Level REST client are:

Maven groupId: `org.elasticsearch.client`

Maven artifactId: `elasticsearch-rest-high-level-client`

Maven version: `7.13.3`

Note

Ensure not to mix the versions in the jar files dependency stack for the Elasticsearch High Level REST Client. Mixing versions results in dependency conflicts.

10.2.20.2 Elasticsearch 8x

The Elasticsearch Handler allows you to store, search, and analyze large volumes of data quickly and in near real time.

This article describes how to use the Elasticsearch handler (starting Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 21.10.0.0.0). In Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) version 21.10.0.0, the Elasticsearch handler was modified to support a new Elasticsearch client. The new client supports Elasticsearch 8.x.

- [Overview](#)
- [Detailing the Functionality](#)
- [About the Index](#)
- [About the Document](#)
- [About the Data Types](#)
- [About the Connection](#)
- [About Supported Operation](#)
- [About DDL Handling](#)
- [About the Primary Key Update](#)
- [About UPSERT](#)
- [About Bulk Write](#)
- [About Routing](#)
- [About Request Headers](#)
- [About Java API Client](#)
- [Setting Up the Elasticsearch Handler](#)
- [Elasticsearch Handler Configuration](#)
- [Enabling Security for Elasticsearch](#)
- [Security Configuration for Elasticsearch Cluster](#)
- [Security Configuration for Elasticsearch Handler](#)
- [Troubleshooting](#)
- [Elasticsearch Handler Client Dependencies](#)

The Elasticsearch cluster must be accessed in secured manner in production environment. Security features must be first enabled in Elasticsearch cluster and those security configurations must be added to Elasticsearch handler properties file

The latest version of Elasticsearch has the security auto-configured when it is installed and started. The logs will print security details for auto-configured cluster as follows:

What are the dependencies for the Elasticsearch Handler to connect to Elasticsearch databases?

10.2.20.2.1 Overview

Elasticsearch is a highly scalable open-source full-text search and analytics engine. Elasticsearch allows you to store, search, and analyze large volumes of data quickly and in

near real time. It is generally used as the underlying engine or technology that drives applications with complex search features.

The Elasticsearch Handler uses the Elasticsearch Java client to connect and receive data into Elasticsearch node, see <https://www.elastic.co>.

10.2.20.2.2 Detailing the Functionality

This topic details the Elasticsearch Handler functionality.

10.2.20.2.3 About the Index

An Elasticsearch **index** is a collection of documents with similar characteristics. An index can only be created in lowercase. An Elasticsearch **type** is a logical group within an index. All the documents within an index or type should have same number and type of fields. Index in Elasticsearch is equivalent to table in RDBMS.

For three-part table names in source trail, the index is constructed by concatenating source catalog, schema, and table name. The Elasticsearch Handler maps the source trail schema concatenated with source trail table name to construct the index when there is no catalog in source table.

Table 10-21 Elasticsearch Mapping

Source Trail	Elasticsearch Index
schema.tablename	schema_tablename
catalog.schema.tablename	catalog_schema_tablename

If an index does not already exist in the Elasticsearch cluster, a new index is created when Elasticsearch Handler receives (INSERT or UPDATE operation in source trail) data.

If Handler receives DELETE operation in source trail but the index does not exist in Elasticsearch cluster, then the handler will ABEND.

10.2.20.2.4 About the Document

An Elasticsearch document is a basic unit of information that can be indexed. Within an index or type, you can store as many documents as you want. Each document has a unique identifier based on the `_id` field.

If Handler receives DELETE operation in source trail but the index does not exist in Elasticsearch cluster, then the handler will ABEND.

10.2.20.2.5 About the Data Types

Elasticsearch supports the following data types:

- 32-bit integer
- 64-bit integer
- Double
- Date
- String
- Binary

10.2.20.2.6 About the Connection

A **cluster** is a collection of one or more nodes (servers) that holds the entire data. It provides federated indexing and search capabilities across all nodes.

A **node** is a single server that is part of the cluster, stores the data, and participates in the cluster's indexing and searching.

The Elasticsearch Handler property `gg.handler.name.ServerAddressList` can be set to point to the nodes available in the cluster.

Elasticsearch Handler uses the Java API client to connect to Elasticsearch cluster nodes configured in above handler property via http/https protocol, even though the cluster nodes internally communicate with each other using transport layer protocol.

Port for http/https must be configured in handler property (instead of transport port) for connection via Elasticsearch client.

10.2.20.2.7 About Supported Operation

The Elasticsearch Handler supports the following operations for replication to Elasticsearch cluster in the target.

INSERT

The Elasticsearch Handler creates a new index if the index does not exist, and then inserts a new document. If the `_id` is already present, it overwrites (replaces) the existing record with new record with same `_id`.

UPDATE

If an Elasticsearch index or document exists, the document is updated. If an Elasticsearch index or document does not exist, then a new index is created and the column values in the UPDATE operation are inserted as a new document.

DELETE

If an Elasticsearch index or `_id` of document exists, then the document is deleted. If `_id` of document does not exist, then it continues without doing anything. If Elasticsearch index is missing, then it will ABEND the handler.

The TRUNCATE operation is not supported.

10.2.20.2.8 About DDL Handling

The Elasticsearch Handler does not react to any DDL records in the source trail. Any data manipulation records for a new source table results in auto-creation of index or type in the Elasticsearch cluster.

10.2.20.2.9 About the Primary Key Update

The Elasticsearch document identifier is created based on the source table's primary key column value. The document identifier cannot be modified.

The Elasticsearch handler processes a source primary key's update operation by performing a DELETE followed by an INSERT. While performing the INSERT, there is a possibility that the new document may contain fewer fields than required.

For the INSERT operation to contain all the fields in the source table, enable trail Extract to capture the full data before images for update operations or use GETBEFORECOLS to write the required column's before images.

10.2.20.2.10 About UPSERT

The Elasticsearch handler supports UPSERT mode for UPDATE operations. This mode can be enabled by setting the Elasticsearch handler property `gg.handler.name.upsert` as `true`. This is enabled by default.

The UPSERT mode ensures that for an UPDATE operation from source trail, if the index or the `_id` of document is missing from Elasticsearch cluster, it will create the index and convert the operation to INSERT for adding it as a new record.

Elasticsearch Handler will ABEND for same scenario when UPSERT is `false`.

In future releases, this mechanism will be enhanced to be in line with HANDLECOLLISION mode Oracle GoldenGate where:

- An insert collision should result in duplicate error.
- A missing update or delete should result in not found error.

The corresponding error codes will be returned back to replicat and handled by it as per Oracle GoldenGate handle collision strategy.

10.2.20.2.11 About Bulk Write

The Elasticsearch handler supports bulk operation mode where multiple operations can be grouped into a batch and whole batch can be applied to target Elasticsearch cluster in one shot. This improves the performance.

Bulk mode can be enabled by setting the value of Elasticsearch handler property `gg.handler.name.bulkWrite` as `true`. It is disabled by default.

Bulk mode has a few limitations. If there is any failure (exception thrown) for an operation in bulk, it can result in inconsistent data at target. For example, a delete operation where the index is missing from the target Elasticsearch cluster, it will result in exception. If such an operation is part of a batch in bulk mode, then the batch is not applied after the failure of that operation, resulting in inconsistency.

To avoid bulk API errors, you must set the handler `MAXTRANSOPS` size to match the bulk thread pool queue size at a minimum.

The configuration `thread_pool.bulk.queue_size` property can be modified in the `elasticsearch.yaml` file.

10.2.20.2.12 About Routing

A document is routed to a particular shard in an index using the `_routing` value. The default `_routing` value is the document's `_id` field. Custom routing patterns can be implemented by specifying a custom routing value per document.

Elasticsearch Handler supports custom routing by specifying the mapping field key in the property `gg.handler.name.routingKeyMappingTemplate` of Elasticsearch handler properties file.

10.2.20.2.13 About Request Headers

Elasticsearch allows sending additional request headers (header name and value pair) along with the http requests of REST calls. The Elasticsearch Handler supports sending additional headers by specifying header name and value pairs in the Elasticsearch Handler property `gg.handler.name.headers` in the properties file.

10.2.20.2.14 About Java API Client

Elasticsearch Handler now uses Java API Client to connect Elasticsearch cluster for performing all operations of replication. It internally uses Elasticsearch Rest Client and Transport Client to perform all the operations. The older clients like Rest High-Level Client and Transport Client are deprecated and hence removed.

Supported Versions of Elasticsearch Cluster

To configure this handler, Elasticsearch cluster version 7.16.x or above must be configured and running. To configure Elasticsearch cluster, see [Get Elasticsearch up and running](#)

10.2.20.2.15 Setting Up the Elasticsearch Handler

You must ensure that the Elasticsearch cluster is setup correctly and the cluster is up and running. Supported versions of Elasticsearch cluster are 7.16.x and above. See <https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>. Alternatively, you can use Kibana to verify the setup.

10.2.20.2.16 Elasticsearch Handler Configuration

To configure the Elasticsearch Handler, the parameter file (`res.prm`) and the properties (`elasticsearch.props`) file must be configured with valid values.

Parameter File:

Parameter file should point to the correct properties file for Elasticsearch Handler.

The following are the mandatory parameters for parameter file (`res.prm`) necessary for running Elasticsearch Handler:

- REPLICAT replicat-name
- TARGETDB LIBFILE libggjava.so SET property=dirprm/elasticsearch.props
- MAP schema-name.table-name, TARGET schema-name.table-name

Properties File:

The following are the mandatory properties for properties file (`elasticsearch.props`), which is necessary for running Elasticsearch handler:

- gg.handlerlist=elasticsearch
- gg.handler.elasticsearch.type=elasticsearch
- gg.handler.elasticsearch.ServerAddressList=127.0.0.1:9200

Table 10-22 Elasticsearch Handler Configuration Properties

Property Name	Required (Yes/No)	Legal Values (Default value)	Explanation
gg.handler.name.ServerAddressList	Yes	[<Hostname ip>:<port>,<Hostname ip>:<port>, ...] [localhost:9200]	List of valid hostnames (or IP) and port number separated by ‘.’ of cluster nodes of Elasticsearch cluster.
gg.handler.name.BulkWrite	No	[true false] Default [false]	If Bulk Write mode is enabled (set true), the operations of transaction will be stored in batch and applied to target ES cluster in one shot for a batch (transaction) depending on batch size.
gg.handler.name.Upsert	No	[true false] [true]	If upsert mode is enabled (set to true), the update operation will be inserted as new document when it's missing on target ES cluster.
gg.handler.name.NumberAsString	No	[true false] [false]	Set if the number will be stored as string.
gg.handler.name.ProxyServer	No	[Proxy-Hostname Proxy-IP]	Proxy server hostname (or IP) to connect to Elasticsearch cluster.
gg.handler.name.ProxyPort	No	[Port number]	Port number of proxy server. Required if proxy is configured.
gg.handler.name.ProxyProtocol	No	[http https] [http]	Protocol for Proxy server connection.
gg.handler.name.ProxyUsername	No	[Username of proxy server]	Username for connecting to Proxy server.
gg.handler.name.ProxyPassword	No	[Password of proxy server]	Password for connecting to Proxy server. This can be encrypted using ORACLEWALLET.
gg.handler.name.AuthType	No	[basic ssl none] [none]	Authentication type to be used for connecting to Elasticsearch cluster.
gg.handler.name.BasicAuthUsername	No	[username of ES cluster]	Username credential for basic authentication to connect ES server. This can be encrypted using ORACLEWALLET.
gg.handler.name.BasicAuthPassword	No	[password of ES cluster]	Password credential for basic authentication to connect ES server. This can be encrypted using ORACLEWALLET.

Table 10-22 (Cont.) Elasticsearch Handler Configuration Properties

Property Name	Required (Yes/No)	Legal Values (Default value)	Explanation
gg.handler.name.Fingerprint	No	[fingerprint hash code]	It is the hash of a certificate calculated on all certificate's data and its signature. Applicable for authentication type SSL. This can be encrypted using ORACLEWALLET.
gg.handler.name.CertFilePath	No	[/path/to/CA_certificate_file.crt]	CA certificate file (.crt) for SSL/TLS authentication.
gg.handler.name.TrustStore	No	[/Path/to/trust-store-file]	Path to Trust-store file in server for SSL / TLS server authentication. Applicable for authentication type SSL.
gg.handler.name.TrustStorePassword	No	[trust-store password]	Password for Trust-store file for SSL/TLS authentication. Applicable for authentication type SSL. This can be encrypted using ORACLEWALLET.
gg.handler.name.TrustStoreType	No	[jks pkcs12] [jks]	The key-store type for SSL/TLS authentication. Applicable if authentication type is SSL.
gg.handler.name.RoutingKeyMappingTemplate	No	[Routing field-name]	This defines the field-name whose value will be mapped for routing to particular shard in an index of ES cluster.
gg.handler.name.Headers	No	[<key>:<value>, <key>:<value>, ...]	List of name and value pair of headers to be sent with REST calls.
gg.handler.name.MaxConnectTimeout	No	Time in seconds	Time in seconds that request will wait for connecting to Elasticsearch server.
gg.handler.name.MaxSocketTimeout	No	Time in seconds	Time in seconds that request will wait for response to come from Elasticsearch server.
gg.handler.name.IOThreadCount	No	Count	Count of thread to handle IO requests.

Table 10-22 (Cont.) Elasticsearch Handler Configuration Properties

Property Name	Required (Yes/No)	Legal Values (Default value)	Explanation
gg.handler.name.NodeSelector	No	ANY SKIP_DEDICATED_MASTERS [Fully qualified name of node selector class] [ANY]	Predefined strategy ANY or SKIP_DEDICATED_MASTERS. Or fully qualified name of class that implements custom strategy (by implementing NodeSelector.java interface).

Set the Classpath

The Elasticsearch handler property `gg.classpath` must include all the dependency jars required by the Java API client. For a listing and downloading of the required client JAR files, use the Dependency Downloader script `elasticsearch_java.sh` in `OGG_HOME/DependencyDownloader` directory and pass the version 8.7.0 as argument. For more information about Elasticsearch client dependencies, see [Elasticsearch Handler Client Dependencies](#).

It creates a directory `OGG_HOME/DependencyDownloader/dependencies/elasticsearch_rest_8.7.0` and downloads all the dependency jars inside it. The client library version 8.7.0 can be used for all supported Elasticsearch clusters.

This location can be configured in classpath as: `gg.classpath=/path/to/OGG_HOME/DependencyDownloader/dependencies/elasticsearch_rest_8.7.0/*`

The inclusion of the `*` wildcard character at the end of the path can be used in order to include all of the JAR files in that directory in the associated classpath. Do not use `*.jar`.

Sample Configuration of Elasticsearch Handler:

For reference, to configure Elasticsearch handler, sample parameter (`res.prm`) and sample properties file (`elasticsearch.props`) for Elasticsearch handler is available in directory:

`OGG_HOME/AdapterExamples/big-data/elasticsearch`

10.2.20.2.17 Enabling Security for Elasticsearch

The Elasticsearch cluster must be accessed in secured manner in production environment. Security features must be first enabled in Elasticsearch cluster and those security configurations must be added to Elasticsearch handler properties file

10.2.20.2.18 Security Configuration for Elasticsearch Cluster

The latest version of Elasticsearch has the security auto-configured when it is installed and started. The logs will print security details for auto-configured cluster as follows:

- Elasticsearch security features have been automatically configured!
- Authentication is enabled and cluster connections are encrypted.
- Password for the elastic user (reset with ``bin/elasticsearch-reset-password -u elastic``): <generated_password>

- HTTP CA certificate SHA-256 fingerprint: <certificate_fingerprint>
- Configure Kibana to use this cluster:
- Run Kibana and click the configuration link in the terminal when Kibana starts.
- Copy the following enrollment token and paste it into Kibana in your browser (valid for the next 30 minutes): <enrollment_token>

These security parameter values must be noted down and used to configure Elasticsearch handler. All the auto-generated certificates are created inside ElasticSearch-install-directory/config/cert folder.

If security is not auto-configured for older versions of Elasticsearch, we need to manually enable the security features like basic and encrypted (SSL) authentication in below configuration file of Elasticsearch cluster before running it.

Elasticsearch-installation-directory/config/elasticsearch.yml

Following parameters must be added to enable security features in elasticsearch.yml file and restarting the Elasticsearch cluster.

```
#----- BEGIN SECURITY AUTO CONFIGURATION -----
# The following settings, TLS certificates and keys have been
# configured for SSL/TLS authentication.
# -----
# Enable security features
xpack.security.enabled: true
xpack.security.enrollment.enabled: true

# Enable encryption for HTTP API client connections
xpack.security.http.ssl:
  enabled: true
  keystore.path: certs/http.p12

# Enable encryption and mutual authentication between cluster nodes
xpack.security.transport.ssl:
  enabled: true
  verification_mode: certificate
  keystore.path: certs/transport.p12
  truststore.path: certs/transport.p12
# Create a new cluster with the current node only
# Additional nodes can still join the cluster later
cluster.initial_master_nodes: ["cluster-host-name"]

# Allow HTTP API connections from anywhere
# Connections are encrypted and require user authentication
http.host: 0.0.0.0
#----- END SECURITY AUTO CONFIGURATION -----
```

For more information about the security setting of Elasticsearch cluster, see <https://www.elastic.co/guide/en/elasticsearch/reference/current/manually-configure-security.html>

10.2.20.2.19 Security Configuration for Elasticsearch Handler

Elasticsearch handler supports three modes of security configuration which can be configured using the Elasticsearch Handler property `gg.handler.name.authType` with following values: `Elasticsearch-installation-directory/config/elasticsearch.yml`

1. **None:** This mode is used when no security feature is enabled in Elasticsearch stack. No other configuration is required for this mode and Elasticsearch can be accessed directly using http protocol.
2. **Basic:** This mode is used when only basic security feature is enabled for a user by setting a username and password for the user. The basic authentication username and password property must be provided in properties file in order to access the Elasticsearch cluster.

```
gg.handler.name.authType=basic
gg.handler.name.basicAuthUsername=elastic
gg.handler.name.basicAuthPassword=changeme
```

3. **SSL:** This mode is used when SSL/TLS authentication is configured for encryption in Elasticsearch stack. User must provide either of CA fingerprint hash, path to CA certificate file (.crt) OR path to trust-store file (along with trust-store type and trust-store password) for handler to be able to connect to Elasticsearch cluster. This mode also supports combination of SSL/TLS authentication and Basic authentication configured in Elasticsearch stack. User must configure both basic authentication properties (username and password) and SSL related properties (fingerprint or certificate file or trust-store), if both are configured in Elasticsearch cluster.

```
gg.handler.name.authType=ssl

# if basic authentication username and password is configured.
gg.handler.name.basicAuthUsername=username
gg.handler.name.basicAuthPassword=password

# for SSL one of these three must be configured
gg.handler.name.certFilePath=/path/to/ESHome/config/certs/http_ca.crt
OR
gg.handler.name.fingerprint=862e3f117c386a63f8f43db88760d463900e4c814590b89
20e1c0e25f6db4df4
OR
gg.handler.name.trustStore=/path/to/http.p12
gg.handler.name.trustStoreType=pkcs12
gg.handler.name.trustStorePassword=pass
```

All the above security related properties that contains confidential information can be configured to use Oracle Wallet for encrypting their confidential values in properties file.

10.2.20.2.20 Troubleshooting

1. **Error:** `org.elasticsearch.ElasticsearchException[Index [index-name] is not found]` - This exception occurs when there is a delete operation and the corresponding index of delete operation is not present in the Elasticsearch cluster. This can also occur for the update operation if `upsert=false` and the index is missing.

2. **Error:** javax.net.ssl.SSLHandshakeException: [Connection failed] - This can happen when properties for enabling authentication in the elasticsearch.yml file mentioned above are missing for authentication type SSL.
3. **Error:** javax.net.ssl.SSLException: [Received fatal alert: bad_certificate] - This issue comes when host validation fails. Check that certificates generated using cert-utils in Elasticsearch contains the host information.

10.2.20.2.21 Elasticsearch Handler Client Dependencies

What are the dependencies for the Elasticsearch Handler to connect to Elasticsearch databases?

The maven central repository artifacts for Elasticsearch databases are:

Maven groupId: co.elastic.clients

Maven artifactId: elasticsearch-java

Version: 8.7.0

- [Elasticsearch 8.7.0](#)

10.2.20.2.21.1 Elasticsearch 8.7.0

```
commons-codec-1.15.jar
commons-logging-1.2.jar
elasticsearch-java-8.7.0.jar
elasticsearch-rest-client-8.7.0.jar
httpasyncclient-4.1.5.jar
httpclient-4.5.13.jar
httpcore-4.4.13.jar
httpcore-nio-4.4.13.jar
jakarta.json-api-2.0.1.jar
jsr305-3.0.2.jar
parsson-1.0.0.jar
```

10.2.20.3 Support for Vector Data

Elasticsearch handler supports replication of numeric vector / array type data in record by mapping it into the dense vector type field of Elasticsearch. Dense vector is a new data type introduced in Elasticsearch version 8.11.0 to store numeric array of any dimension primarily used for k-nearest neighbor (kNN) search.

See <https://www.elastic.co/guide/en/elasticsearch/reference/current/dense-vector.html>

Note

The automatic creation of indices by Elasticsearch on insertion of records will not map the vector/array data into dense vector type field of Elasticsearch. Index with dense vector type field must be explicitly created to map the vector data into dense vector field of Elasticsearch.

Vector data with different dimensions are not supported by the dense_vector type field in Elasticsearch. It can support fixed dimension vector/array data. Dimension can be explicitly specified while creating the index. If not specified, it will take the dimension of the first record inserted to that field.

10.2.21 Flat Files

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) supports writing data files to a local file system with File Writer Handler.

GG for DAA supports loading data files created by File Writer into Cloud storage services. In these cases, File Writer Handler should be used with one of the following cloud storage configurations:

- [Amazon S3](#)
- [Azure Data Lake Storage](#)
- [Google Cloud Storage](#)
- [Oracle Cloud Infrastructure Object Storage](#)
- [File Writer Handler](#)
You can use the File Writer Handler and the event handlers to transform data.
- [Optimized Row Columnar \(ORC\)](#)
The Optimized Row Columnar (ORC) Event Handler to generate data files is in ORC format.
- [Parquet](#)
Learn how to use the Parquet load files generated by the File Writer Handler into HDFS.

10.2.21.1 File Writer Handler

You can use the File Writer Handler and the event handlers to transform data.

The File Writer Handler supports generating data files in delimited text, XML, JSON, Avro, and Avro Object Container File formats. It is intended to fulfill an extraction, load, and transform use case. Data files are staged on your local file system. Then when writing to a data file is complete, you can use a third party application to read the file to perform additional processing.

The File Writer Handler also supports the event handler framework. The event handler framework allows data files generated by the File Writer Handler to be transformed into other formats, such as Optimized Row Columnar (ORC) or Parquet. Data files can be loaded into third party applications, such as HDFS or Amazon s3. The event handler framework is extensible allowing more event handlers performing different transformations or loading to different targets to be developed. Additionally, you can develop a custom event handler for your Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) environment.

GG for DAA provides two handlers to write to HDFS. Oracle recommends that you use the HDFS Handler or the File Writer Handler in the following situations:

The HDFS Handler is designed to stream data directly to HDFS.

Use when no post write processing is occurring in HDFS. The HDFS Handler does not change the contents of the file, it simply uploads the existing file to HDFS.

Use when analytical tools are accessing data written to HDFS in real time including data in files that are open and actively being written to.

The File Writer Handler is designed to stage data to the local file system and then to load completed data files to HDFS when writing for a file is complete.

Analytic tools are not accessing data written to HDFS in real time.

Post write processing is occurring in HDFS to transform, reformat, merge, and move the data to a final location.

You want to write data files to HDFS in ORC or Parquet format.

- [Detailing the Functionality](#)
- [Configuring the File Writer Handler](#)
- [Stopping the File Writer Handler](#)
- [Review a Sample Configuration](#)
- [File Writer Handler Partitioning](#)
Partitioning functionality had been added to the File Writer Handler in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 21.1. The partitioning functionality uses the template mapper functionality to resolve partitioning strings. The result is that you are afforded control in how to partition source trail data.

10.2.21.1.1 Detailing the Functionality

- [Using File Roll Events](#)
- [Automatic Directory Creation](#)
- [About the Active Write Suffix](#)
- [Maintenance of State](#)

10.2.21.1.1.1 Using File Roll Events

A **file roll event** occurs when writing to a specific data file is completed. No more data is written to that specific data file.

Finalize Action Operation

You can configure the finalize action operation to clean up a specific data file after a successful file roll action using the `finalizeaction` parameter with the following options:

none

Leave the data file in place (removing any active write suffix, see [About the Active Write Suffix](#)).

delete

Delete the data file (such as, if the data file has been converted to another format or loaded to a third party application).

move

Maintain the file name (removing any active write suffix), but move the file to the directory resolved using the `movePathMappingTemplate` property.

rename

Maintain the current directory, but rename the data file using the `fileRenameMappingTemplate` property.

move-rename

Rename the file using the file name generated by the `fileRenameMappingTemplate` property and move the file to the directory resolved using the `movePathMappingTemplate` property.

Typically, event handlers offer a subset of these same actions.

A sample Configuration of a finalize action operation:

```
gg.handlerlist=filewriter
#The File Writer Handler
```

```
gg.handler.filewriter.type=filewriter
gg.handler.filewriter.mode=op
gg.handler.filewriter.pathMappingTemplate=./dirout/evActParamS3R
gg.handler.filewriter.stateFileDirectory=./dirsta
gg.handler.filewriter.fileNameMappingTemplate=${fullyQualifiedTableName}_${currentTimestamp}.txt
gg.handler.filewriter.fileRollInterval=7m
gg.handler.filewriter.finalizeAction=delete
gg.handler.filewriter.inactivityRollInterval=7m
```

File Rolling Actions

Any of the following actions trigger a file roll event.

- A metadata change event.
- The maximum configured file size is exceeded
- The file roll interval is exceeded (the current time minus the time of first file write is greater than the file roll interval).
- The inactivity roll interval is exceeded (the current time minus the time of last file write is greater than the file roll interval).
- The File Writer Handler is configured to roll on shutdown and the Replicat process is stopped.

Operation Sequence

The file roll event triggers a sequence of operations to occur. It is important that you understand the order of the operations that occur when an individual data file is rolled:

1. The active data file is switched to inactive, the data file is flushed, and state data file is flushed.
2. The configured event handlers are called in the sequence that you specified.
3. The finalize action is executed on all the event handlers in the reverse order in which you configured them. Any finalize action that you configured is executed.
4. The finalize action is executed on the data file and the state file. If all actions are successful, the state file is removed. Any finalize action that you configured is executed.

For example, if you configured the File Writer Handler with the Parquet Event Handler and then the S3 Event Handler, the order for a roll event is:

1. The active data file is switched to inactive, the data file is flushed, and state data file is flushed.
2. The Parquet Event Handler is called to generate a Parquet file from the source data file.
3. The S3 Event Handler is called to load the generated Parquet file to S3.
4. The finalize action is executed on the S3 Parquet Event Handler. Any finalize action that you configured is executed.
5. The finalize action is executed on the Parquet Event Handler. Any finalize action that you configured is executed.
6. The finalize action is executed for the data file in the File Writer Handler

10.2.21.1.1.2 Automatic Directory Creation

You do not have to configure write directories before you execute the handler. The File Writer Handler checks to see if the specified write directory exists before creating a file and recursively creates directories as needed.

10.2.21.1.1.3 About the Active Write Suffix

A common use case is using a third party application to monitor the write directory to read data files. Third party application can only read a data file when writing to that file has completed. These applications need a way to determine if writing to a data file is active or complete. The File Writer Handler allows you to configure an **active write suffix** using this property:

```
gg.handler.name.fileWriteActiveSuffix=.tmp
```

The value of this property is appended to the generated file name. When writing to the file is complete, the data file is renamed and the active write suffix is removed from the file name. You can set your third party application to monitor your data file names to identify when the active write suffix is removed.

10.2.21.1.1.4 Maintenance of State

Previously, all Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) handlers have been stateless. These stateless handlers only maintain state in the context of the Replicat process that it was running. If the Replicat process was stopped and restarted, then all the state was lost. With a Replicat restart, the handler began writing with no contextual knowledge of the previous run.

The File Writer Handler provides the ability of maintaining state between invocations of the Replicat process. By default with a restart:

- the state saved files are read,
- the state is restored,
- and appending active data files continues where the previous run stopped.

You can change this default action to require all files be rolled on shutdown by setting this property:

```
gg.handler.name.rollOnShutdown=true
```

10.2.21.1.2 Configuring the File Writer Handler

Lists the configurable values for the File Writer Handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the File Writer Handler, you must first configure the handler type by specifying `gg.handler.name.type=filewriter` and the other File Writer properties as follows:

Table 10-23 File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.type</code>	Required	<code>filewriter</code>	None	Selects the File Writer Handler for use.

Table 10-23 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.maxFileSize</code>	Optional	Default unit of measure is bytes. You can stipulate k, m, or g to signify kilobytes, megabytes, or gigabytes respectively. Examples of legal values include 10000, 10k, 100m, 1.1g.	1g	Sets the maximum file size of files generated by the File Writer Handler. When the file size is exceeded, a roll event is triggered.
<code>gg.handler.name</code> <code>.fileRollInterval</code>	Optional	The default unit of measure is milliseconds. You can stipulate ms, s, m, h to signify milliseconds, seconds, minutes, or hours respectively. Examples of legal values include 10000, 10000ms, 10s, 10m, or 1.5h. Values of 0 or less indicate that file rolling on time is turned off.	File rolling on time is off.	The timer starts when a file is created. If the file is still open when the interval elapses then the a file roll event will be triggered.

Table 10-23 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.inactivityRollInterval</code>	Optional	The default unit of measure is millisecond s. You can stipulate ms, s, m, h to signify millisecond s, seconds, minutes, or hours respectively. Examples of legal values include 10000, 10000ms, 10s, 10m, or 1.5h. Values of 0 or less indicate that file rolling on time is turned off.	File inactivity rolling is turned off.	The timer starts from the latest write to a generated file. New writes to a generated file restart the counter. If the file is still open when the timer elapses a roll event is triggered..
<code>gg.handler.name.fileNameMappingTemplate</code>	Required	A string with resolvable keywords and constants used to dynamically generate File Writer Handler data file names at runtime.	None	Use keywords interlaced with constants to dynamically generate unique file names at runtime. Typically, file names follow the format, <code>/some/path/\${tableName}_\${groupName}_\${currentTimestamp}.txt</code> . See Template Keywords .

Table 10-23 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.pathMappingTemplate</code>	Required	A string with resolvable keywords and constants used to dynamically generate the directory to which a file is written.	None	Use keywords interlaced with constants to dynamically generate unique path names at runtime. Typically, path names follow the format, <code>/some/path/\${tableName}</code> . See Template Keywords .
<code>gg.handler.name</code> <code>.fileWriteActiveSuffix</code>	Optional	A string.	None	An optional suffix that is appended to files generated by the File Writer Handler to indicate that writing to the file is active. At the finalize action the suffix is removed.
<code>gg.handler.name</code> <code>.stateFileDirectory</code>	Required	A directory on the local machine to store the state files of the File Writer Handler.	None	Sets the directory on the local machine to store the state files of the File Writer Handler. The group name is appended to the directory to ensure that the functionality works when operating in a coordinated apply environment.
<code>gg.handler.name</code> <code>.rollOnShutdown</code>	Optional	true false	false	Set to true, on normal shutdown of the Replicat process all open files are closed and a file roll event is triggered. If successful, the File Writer Handler has no state to carry over to a restart of the File Writer Handler.

Table 10-23 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler.name .finalizeAction	Optional	none delete move rename move- rename	none	Indicates what the File Writer Handler should do at the finalize action. none Leave the data file in place (removing any active write suffix, see About the Active Write Suffix). delete Delete the data file (such as, if the data file has been converted to another format or loaded to a third party application). move Maintain the file name (removing any active write suffix), but move the file to the directory resolved using the <code>movePathMappingTemplate</code> property. rename Maintain the current directory, but rename the data file using the <code>fileRenameMappingTemplate</code> property. move - rename Rename the file using the file name generated by the <code>fileRenameMappingTemplate</code> property and move the file to the directory resolved using the <code>movePathMappingTemplate</code> property.
gg.handler.name .partitionByTable	Optional	true false	true	Set to true so that data from different source tables is partitioned into separate files. Set to false to interlace operation data from all source tables into a single output file. It cannot be set to false if the file format is the Avro OCF (Object Container File) format.
gg.handler.name .eventHandler	Optional	HDFS ORC PARQUET S3	No event handler configured.	A unique string identifier cross referencing an event handler. The event handler will be invoked on the file roll event. Event handlers can do thing file roll event actions like loading files to S3, converting to Parquet or ORC format, or loading files to HDFS.

Table 10-23 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.fileRenameMappingTemplate</code>	Required if <code>gg.handler.name.finalize</code> Action is set to rename or move-rename.	A string with resolvable keywords and constants used to dynamically generate File Writer Handler data file names for file renaming in the finalize action.	None.	Use keywords interlaced with constants to dynamically generate unique file names at runtime. Typically, file names follow the format, <code>\${fullyQualifiedTableName}_\${groupName}_\${currentTimestamp}.txt</code> . See Template Keywords .
<code>gg.handler.name</code> <code>.movePathMappingTemplate</code>	Required if <code>gg.handler.name.finalize</code> Action is set to rename or move-rename.	A string with resolvable keywords and constants used to dynamically generate the directory to which a file is written.	None	Use keywords interlaced with constants to dynamically generate a unique path names at runtime. Typically, path names typically follow the format, <code>/ogg/data/\${groupName}/\${fullyQualifiedTableName}</code> . See Template Keywords .

Table 10-23 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler.name .format	Required	delimited text json json_row xml avro_row avro_op avro_row_ ocf avro_op_o cf	delimitedtext	Selects the formatter for the HDFS Handler for how output data will be formatted delimitedtext Delimited text. json JSON json_row JSON output modeling row data xml XML avro_row Avro in row compact format. avro_op Avro in operation more verbose format. avro_row_ocf Avro in the row compact format written into HDFS in the Avro Object Container File (OCF) format. avro_op_ocf Avro in the more verbose format written into HDFS in the Avro OCF format. If you want to use the Parquet or ORC Event Handlers, then the selected format must be avro_row_ocf or avro_op_ocf.
gg.handler.name .bom	Optional	An even number of hex characters.	None	Enter an even number of hex characters where every two characters correspond to a single byte in the byte order mark (BOM). For example, the string efbbbf represents the 3-byte BOM for UTF-8.
gg.handler.name .createControlFile	Optional	true false	false	Set to true to create a control file. A control file contains all of the completed file names including the path separated by a delimiter. The name of the control file is <code>{groupName}.control</code> . For example, if the Replicat process name is fw, then the control file name is <code>FW.control</code> .
gg.handler.name .controlFileDelimiter	Optional	Any string	new line (\n)	Allows you to control the delimiter separating file names in the control file. You can use CDATA[] wrapping with this property.

Table 10-23 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name .controlFileDir ectory</code>	Optional	A path to a directory to hold the control file.	A period (.) or the Oracle GoldenGate installation directory.	Set to specify where you want to write the control file.
<code>gg.handler.name .createOwnerFile</code>	Optional	true false	false	Set to true to create an owner file. The owner file is created when the Replicat process starts and is removed when it terminates normally. The owner file allows other applications to determine if the process is running. The owner file remains in place when the Replicat process ends abnormally. The name of the owner file is the <code>{groupName}.owner</code> . For example, if the replicat process is name <code>fw</code> , then the owner file name is <code>FW.owner</code> . The file is create in the . directory or the Oracle GoldenGate installation directory.
<code>gg.handler.name .atTime</code>	Optional	One or more times to trigger a roll action of all open files.	None	Configure one or more trigger times in the following format: HH:MM, HH:MM, HH:MM Entries are based on a 24 hour clock. For example, an entry to configure rolled actions at three discrete times of day is: <code>gg.handler.fw.atTime=03:30,21:00,23:51</code>
<code>gg.handler.name .avroCodec</code>	Optional	null no compression.	null bzip2 deflate snappy xz	Enables the corresponding compression algorithm for generated Avro OCF files. The corresponding compression library must be added to the <code>gg.classpath</code> when compression is enabled.
<code>gg.handler.name .bufferSize</code>	Optional	1024	Positive Integer >= 512	Sets the size the <code>BufferedOutputStream</code> for each active writestream. Setting to a larger value may improve performance especially when there are a few active write streams, but a large number of operations are being written to those streams. If there are a large number of active write streams, increasing the value with this property is likely undesirable and could result in an out of memory exception by exhausting the Java heap.

Table 10-23 (Cont.) File Writer Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler.name .rollOnTruncate	Optional	true false	false	Controls whether the occurrence of truncate operation causes a rollover of the corresponding data file by the handler. The default is <code>false</code> , which means the corresponding data file is not rolled when a truncate operation is presented. Set to <code>true</code> to roll the data file on a truncate operation. To propagate truncate operations, ensure to set the replicat property <code>GETTRUNCATES</code> .
gg.handler.name .logEventHandlerStatus	Optional	true false	false	When set to <code>true</code> , it logs the status of completed event handlers at the info logging level. Can be used for debugging and troubleshooting of the event handlers.
gg.handler.name .eventHandlerTimeoutMinutes	Optional	Long integer	120	The event handler thread timeout in minutes. The event handler threads spawned by the file writer handler are provided a max execution time to complete their work. If the timeout value is exceeded, then Replicat assumes that the Event handler thread is hung and will ABEND. For stage and merge use cases, Event handler threads may take longer to complete their work. The default value is set to 120 (2 hours).
gg.handler.name .processBacklogOnShutdown	Optional	true false	false	Set to <code>true</code> to force the replicat to process all of the outstanding staged files through the event handler framework. Recommend setting to <code>true</code> for initial load replication to data warehouse targets. Recommend setting to <code>true</code> for simple data format conversion and/or load to cloud storage scenarios. Recommend setting to <code>false</code> for CDC replication to data warehouse targets as merges can take long periods of time.

10.2.21.1.3 Stopping the File Writer Handler

The replicat process running the File Writer Handler should only be stopped normally.

- Force stop should never be executed on the replicat process.
- The Unix kill command should never be used to kill the replicat process.

The File Writer is writing data files and using state files to track the progress and state. File writing is not transactional. Abnormal ending of the replicat process means that the state of the File Writer Handler can become inconsistent. The best practice is to stop the replicat process normally.

An inconsistent state may mean that the replicat process will abend on startup and require manual removal of state files.

The following is a typical error message for inconsistent state:

```
ERROR 2022-07-11 19:05:23.000367 [main]- Failed to
restore state for UUID [d35f117f-ffab-4e60-aa93-f7ef860bf280]
```

```
table name [QASOURCE.TCUSTORD]
data file name [QASOURCE.TCUSTORD_2022-07-11_19-04-27.900.txt]
```

The error means that the data file has been removed from the file system, but that the corresponding .state file has not yet been removed. Three scenarios can generally cause this problem:

- The replicat process was force stopped, was killed using the kill command, or crashed while it was in the processing window between when the data file was removed and when the associated .state file was removed.
- The user has manually removed the data file or files but left the associated .state file in place.
- There are two instances of the same replicat process running. A lock file is created to prevent this, but there is a window on replicat startup which allows multiple instances of a replicat process to be started.

If this problem occurs, then you should manually determine whether or not the data file associated with the .state file has been successfully processed. If the data has been successfully processed, then you can manually remove the .state file and restart the replicat process.

If data file associated with the problematic .state file has been determined not to have been processed, then do the following:

1. Delete all the .state files.
2. Alter the seqno and rba of the replicat process to back it up to a period for which it was known that processing successfully occurred.
3. Restart the replicat process to reprocess the data.

10.2.21.1.4 Review a Sample Configuration

This File Writer Handler configuration example is using the Parquet Event Handler to convert data files to Parquet, and then for the S3 Event Handler to load Parquet files into S3:

```
gg.handlerlist=filewriter

#The handler properties
gg.handler.name.type=filewriter
gg.handler.name.mode=op
gg.handler.name.pathMappingTemplate=./dirout
gg.handler.name.stateFileDirectory=./dirsta
gg.handler.name.fileNameMappingTemplate=${fullyQualifiedTableName}_${currentTimestamp}.txt
gg.handler.name.fileRollInterval=7m
gg.handler.name.finalizeAction=delete
gg.handler.name.inactivityRollInterval=7m
gg.handler.name.format=avro_row_ocf
gg.includeggtokens=true
gg.handler.name.partitionByTable=true
gg.handler.name.eventHandler=parquet
gg.handler.name.rollOnShutdown=true

gg.eventhandler.parquet.type=parquet
gg.eventhandler.parquet.pathMappingTemplate=./dirparquet
gg.eventhandler.parquet.writeToHDFS=false
gg.eventhandler.parquet.finalizeAction=delete
gg.eventhandler.parquet.eventHandler=s3
gg.eventhandler.parquet.fileNameMappingTemplate=${tableName}_${currentTimestamp}.parquet
```

```
gg.handler.filewriter.eventHandler=s3
gg.eventhandler.s3.type=s3
gg.eventhandler.s3.region=us-west-2
gg.eventhandler.s3.proxyServer=www-proxy.us.oracle.com
gg.eventhandler.s3.proxyPort=80
gg.eventhandler.s3.bucketMappingTemplate=tomsfunbucket
gg.eventhandler.s3.pathMappingTemplate=thepath
gg.eventhandler.s3.finalizeAction=none
```

10.2.21.1.5 File Writer Handler Partitioning

Partitioning functionality had been added to the File Writer Handler in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 21.1. The partitioning functionality uses the template mapper functionality to resolve partitioning strings. The result is that you are afforded control in how to partition source trail data.

All of the keywords that are supported by the templating functionality are now supported in File Writer Handler partitioning.

- [File Writer Handler Partitioning Precondition](#)
In order to use the partitioning functionality, data must first be partitioned by table. The following configuration cannot be set: `gg.handler.filewriter.partitionByTable=false`.
- [Path Configuration](#)
Assume that the path mapping template is configured as follows:
`gg.handler.filewriter.pathMappingTemplate=/ogg/${fullyQualifiedTableName}`. At runtime the path resolves as follows for the `DBO.ORDERS` source table: `/ogg/DBO.ORDERS`.
- [Partitioning Configuration](#)
Any of the keywords that are legal for templating are now legal for partitioning:
`gg.handler.filewriter.partitionner.fully qualified table name=templating keywords and/or constants`.
- [Partitioning Effect on Event Handler](#)
The resolved partitioning path is carried forward to the corresponding Event Handlers as well.

10.2.21.1.5.1 File Writer Handler Partitioning Precondition

In order to use the partitioning functionality, data must first be partitioned by table. The following configuration cannot be set: `gg.handler.filewriter.partitionByTable=false`.

10.2.21.1.5.2 Path Configuration

Assume that the path mapping template is configured as follows:
`gg.handler.filewriter.pathMappingTemplate=/ogg/${fullyQualifiedTableName}`. At runtime the path resolves as follows for the `DBO.ORDERS` source table: `/ogg/DBO.ORDERS`.

10.2.21.1.5.3 Partitioning Configuration

Any of the keywords that are legal for templating are now legal for partitioning:
`gg.handler.filewriter.partitionner.fully qualified table name=templating keywords and/or constants`.

See [Template Keywords](#).

Example 1

Partitioning for the DB0.ORDERS table is set to the following:

```
gg.handler.filewriter.partitioner.DB0.ORDERS=par_sales_region=$  
{columnValue[SALES_REGION]}
```

This example can result in the following breakdown of files on the file system:

```
/ogg/DB0.ORDERS/par_sales_region=west/data files  
/ogg/DB0.ORDERS/par_sales_region=east/data files  
/ogg/DB0.ORDERS/par_sales_region=north/data files  
/ogg/DB0.ORDERS/par_sales_region=south/data file
```

Example 2

Partitioning for the DB0.ORDERS table is set to the following:

```
gg.handler.filewriter.partitioner.DB0.ORDERS=par_sales_region=$  
{columnValue[SALES_REGION]}/par_state=${columnValue[STATE]}
```

This example can result in the following breakdown of files on the file system:

```
/ogg/DB0.ORDERS/par_sales_region=west/par_state=CA/data files  
/ogg/DB0.ORDERS/par_sales_region=east/par_state=FL/data files  
/ogg/DB0.ORDERS/par_sales_region=north/par_state=MN/data files  
/ogg/DB0.ORDERS/par_sales_region=south/par_state=TX/data files
```

Caution

Ensure to be extra vigilant while configuring partitioning. Choosing partitioning column values that have a very large range of data values result in partitioning to a proportional number of output data files.

10.2.21.1.5.4 Partitioning Effect on Event Handler

The resolved partitioning path is carried forward to the corresponding Event Handlers as well.

Example 1

If partitioning is configured as follows:

```
gg.handler.filewriter.partitioner.DB0.ORDERS=par_sales_region=$  
{columnValue[SALES_REGION]}, then the partition string might resolve to the following:
```

```
par_sales_region=west  
par_sales_region=east  
par_sales_region=north  
par_sales_region=south
```

Example 2

If S3 Event handler is used, then the path mapping template of the S3 Event Handler is configured as follows: `gg.eventhandler.s3.pathMappingTemplate=output/dir`. The target directories in S3 are as follows:

```
output/dir/par_sales_region=west/data files  
output/dir/par_sales_region=east/data files  
output/dir/par_sales_region=north/data files  
output/dir/par_sales_region=south/data files
```


10.2.21.2 Optimized Row Columnar (ORC)

The Optimized Row Columnar (ORC) Event Handler to generate data files is in ORC format.

This topic describes how to use the ORC Event Handler.

- [Overview](#)
- [Detailing the Functionality](#)
- [Configuring the ORC Event Handler](#)
- [Optimized Row Columnar Event Handler Client Dependencies](#)

What are the dependencies for the Optimized Row Columnar (OCR) Handler?

10.2.21.2.1 Overview

ORC is a row columnar format that can substantially improve data retrieval times and the performance of Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) analytics. You can use the ORC Event Handler to write ORC files to either a local file system or directly to HDFS. For information, see <https://orc.apache.org/>.

10.2.21.2.2 Detailing the Functionality

- [About the Upstream Data Format](#)
- [About the Library Dependencies](#)
- [Requirements](#)

10.2.21.2.2.1 About the Upstream Data Format

The ORC Event Handler can only convert Avro Object Container File (OCF) generated by the File Writer Handler. The ORC Event Handler cannot convert other formats to ORC data files. The format of the File Writer Handler must be `avro_row_ocf` or `avro_op_ocf`, see [Flat Files](#).

10.2.21.2.2.2 About the Library Dependencies

Generating ORC files requires both the Apache ORC libraries and the HDFS client libraries, see [Optimized Row Columnar Event Handler Client Dependencies](#) and [HDFS Handler Client Dependencies](#).

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) does not include the Apache ORC libraries nor does it include the HDFS client libraries. You must configure the `gg.classpath` variable to include the dependent libraries.

10.2.21.2.2.3 Requirements

The ORC Event Handler can write ORC files directly to HDFS. You must set the `writeToHDFS` property to `true`:

```
gg.eventhandler.orc.writeToHDFS=true
```

Ensure that the directory containing the `HDFS_core-site.xml` file is in `gg.classpath`. This is so the `core-site.xml` file can be read at runtime and the connectivity information to HDFS can be resolved. For example:

```
gg.classpath={HDFS_install_directory}/etc/hadoop
```

If you enable Kerberos authentication is on the HDFS cluster, you have to configure the Kerberos principal and the location of the keytab file so that the password can be resolved at runtime:

```
gg.eventHandler.name.kerberosPrincipal=principal
gg.eventHandler.name.kerberosKeytabFile=path_to_the_keytab_file
```

10.2.21.2.3 Configuring the ORC Event Handler

You configure the ORC Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

The ORC Event Handler works only in conjunction with the File Writer Handler.

To enable the selection of the ORC Handler, you must first configure the handler type by specifying `gg.eventHandler.name.type=orc` and the other ORC properties as follows:

Table 10-24 ORC Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventHandler.name.type</code>	Required	ORC	None	Selects the ORC Event Handler.
<code>gg.eventHandler.name.writeToHDFS</code>	Optional	true false	false	The ORC framework allows direct writing to HDFS. Set to <code>false</code> to write to the local file system. Set to <code>true</code> to write directly to HDFS.
<code>gg.eventHandler.name.pathMappingTemplate</code>	Required	A string with resolvable keywords and constants used to dynamically generate the path in the ORC bucket to write the file.	None	Use keywords interlaced with constants to dynamically generate unique ORC path names at runtime. Typically, path names follow the format, <code>/ogg/data/\${groupName}/\${fullyQualifiedTableName}</code> . See Template Keywords .
<code>gg.eventHandler.name.fileMappingTemplate</code>	Optional	A string with resolvable keywords and constants used to dynamically generate the ORC file name at runtime.	None	Use resolvable keywords and constants used to dynamically generate the ORC data file name at runtime. If not set, the upstream file name is used. See Template Keywords .
<code>gg.eventHandler.name.compressionCodec</code>	Optional	LZ4 LZ0 NONE SNAPPY ZLIB	NONE	Sets the compression codec of the generated ORC file.

Table 10-24 (Cont.) ORC Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler. .name.finalizeA ction	Optional	none delete	none	Set to none to leave the ORC data file in place on the finalize action. Set to delete if you want to delete the ORC data file with the finalize action.
gg.eventhandler. .name.kerberosP rincipal	Optional	The Kerberos principal name.	None	Sets the Kerberos principal when writing directly to HDFS and Kerberos authentication is enabled.
gg.eventhandler. .name.kerberosK eytabFile	Optional	The path to the Keberos keytab file.	none	Sets the path to the Kerberos keytab file with writing directly to HDFS and Kerberos authentication is enabled.
gg.eventhandler. .name.blockPadd ing	Optional	true false	true	Set to true to enable block padding in generated ORC files or false to disable.
gg.eventhandler. .name.blockSize	Optional	long	The ORC default.	Sets the block size of generated ORC files.
gg.eventhandler. .name.bufferSiz e	Optional	integer	The ORC default.	Sets the buffer size of generated ORC files.
gg.eventhandler. .name.encodingS trategy	Optional	COMPRESSI ON SPEED	The ORC default.	Set if the ORC encoding strategy is optimized for compression or for speed..
gg.eventhandler. .name.paddingTo lerance	Optional	A percentage represented as a floating point number.	The ORC default.	Sets the percentage for padding tolerance of generated ORC files.
gg.eventhandler. .name.rowIndexS tride	Optional	integer	The ORC default.	Sets the row index stride of generated ORC files.
gg.eventhandler. .name.stripeSiz e	Optional	integer	The ORC default.	Sets the stripe size of generated ORC files.
gg.eventhandler. .name.eventHand ler	Optional	A unique string identifier cross referencing a child event handler.	No event handler configur ed.	The event handler that is invoked on the file roll event. Event handlers can do file roll event actions like loading files to S3 or HDFS.

Table 10-24 (Cont.) ORC Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler. .name.bloomFilterFpp	Optional	The false positive probability must be greater than zero and less than one. For example, .25 and .75 are both legal values, but 0 and 1 are not.	The Apache ORC default.	Sets the false positive probability of the querying of a bloom filter index and the result indicating that the value being searched for is in the block, but the value is actually not in the block. needs to set which tables to set bloom filters and on which columns. The user selects on which tables and columns to set bloom filters with the following configuration syntax: gg.eventhandler.orc.bloomFilter.QASOURCE.TCUSTMER=CUST_CODE gg.eventhandler.orc.bloomFilter.QASOURCE.TCUSTORD=CUST_CODE,ORDER_DATE QASOURCE.TCUSTMER and QASOURCE.TCUSTORD are the fully qualified names of the source tables. The configured values are one or more columns on which to configure bloom filters. The columns names are delimited by a comma.
gg.eventhandler. .name.bloomFilterVersion	Optional	ORIGINAL UTF8	ORIGIN AL	Sets the version of the ORC bloom filter.

10.2.21.2.4 Optimized Row Columnar Event Handler Client Dependencies

What are the dependencies for the Optimized Row Columnar (OCR) Handler?

The maven central repository artifacts for ORC are:

Maven groupId: org.apache.orc

Maven artifactId: orc-core

Maven version: 1.6.9

The Hadoop client dependencies are also required for the ORC Event Handler, see [Hadoop Client Dependencies](#).

- [ORC Client 1.6.9](#)
- [ORC Client 1.5.5](#)
- [ORC Client 1.4.0](#)

10.2.21.2.4.1 ORC Client 1.6.9

```
aircompressor-0.19.jar
annotations-17.0.0.jar
commons-lang-2.6.jar
commons-lang3-3.12.0.jar
hive-storage-api-2.7.1.jar
jaxb-api-2.2.11.jar
orc-core-1.6.9.jar
```

orc-shims-1.6.9.jar
protobuf-java-2.5.0.jar
slf4j-api-1.7.5.jar
threeten-extra-1.5.0.jar

10.2.21.2.4.2 ORC Client 1.5.5

aircompressor-0.10.jar
asm-3.1.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.1.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-httpclient-3.1.jar
commons-io-2.1.jar
commons-lang-2.6.jar
commons-logging-1.1.1.jar
commons-math-2.1.jar
commons-net-3.1.jar
guava-11.0.2.jar
hadoop-annotations-2.2.0.jar
hadoop-auth-2.2.0.jar
hadoop-common-2.2.0.jar
hadoop-hdfs-2.2.0.jar
hive-storage-api-2.6.0.jar
jackson-core-asl-1.8.8.jar
jackson-mapper-asl-1.8.8.jar
jaxb-api-2.2.11.jar
jersey-core-1.9.jar
jersey-server-1.9.jar
jsch-0.1.42.jar
log4j-1.2.17.jar
orc-core-1.5.5.jar
orc-shims-1.5.5.jar
protobuf-java-2.5.0.jar
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
xmlenc-0.52.jar
zookeeper-3.4.5.jar

10.2.21.2.4.3 ORC Client 1.4.0

aircompressor-0.3.jar
apacheds-i18n-2.0.0-M15.jar
apacheds-kerberos-codec-2.0.0-M15.jar
api-asn1-api-1.0.0-M20.jar
api-util-1.0.0-M20.jar
asm-3.1.jar
commons-beanutils-core-1.8.0.jar
commons-cli-1.2.jar
commons-codec-1.4.jar
commons-collections-3.2.2.jar
commons-compress-1.4.1.jar
commons-configuration-1.6.jar
commons-httpclient-3.1.jar
commons-io-2.4.jar
commons-lang-2.6.jar
commons-logging-1.1.3.jar
commons-math3-3.1.1.jar
commons-net-3.1.jar
curator-client-2.6.0.jar

```
curator-framework-2.6.0.jar  
gson-2.2.4.jar  
guava-11.0.2.jar  
hadoop-annotations-2.6.4.jar  
hadoop-auth-2.6.4.jar  
hadoop-common-2.6.4.jar  
hive-storage-api-2.2.1.jar  
htrace-core-3.0.4.jar  
httpclient-4.2.5.jar  
httpcore-4.2.4.jar  
jackson-core-asl-1.9.13.jar  
jdk.tools-1.6.jar  
jersey-core-1.9.jar  
jersey-server-1.9.jar  
jsch-0.1.42.jar  
log4j-1.2.17.jar  
netty-3.7.0.Final.jar  
orc-core-1.4.0.jar  
protobuf-java-2.5.0.jar  
slf4j-api-1.7.5.jar  
slf4j-log4j12-1.7.5.jar  
xmlenc-0.52.jar  
xz-1.0.jar  
zookeeper-3.4.6.jar
```

10.2.21.3 Parquet

Learn how to use the Parquet load files generated by the File Writer Handler into HDFS.

See [Flat Files](#).

- [Parquet Handler](#)
- [Detailing the Functionality](#)
- [Configuring the Parquet Event Handler](#)
- [Parquet Event Handler Client Dependencies](#)
What are the dependencies for the Parquet Event Handler?

10.2.21.3.1 Parquet Handler

The Parquet Event Handler enables you to generate data files in Parquet format. Parquet files can be written to either the local file system or directly to HDFS. Parquet is a columnar data format that can substantially improve data retrieval times and improve the performance of Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) analytics, see <https://parquet.apache.org/>.

10.2.21.3.2 Detailing the Functionality

- [Configuring the Parquet Event Handler to Write to HDFS](#)
- [About the Upstream Data Format](#)

10.2.21.3.2.1 Configuring the Parquet Event Handler to Write to HDFS

The Apache Parquet framework supports writing directly to HDFS. The Parquet Event Handler can write Parquet files directly to HDFS. These additional configuration steps are required:

The Parquet Event Handler dependencies and considerations are the same as the HDFS Handler, see [HDFS Additional Considerations](#).

Set the `writeToHDFS` property to `true`:

```
gg.eventhandler.parquet.writeToHDFS=true
```

Ensure that `gg.classpath` includes the HDFS client libraries.

Ensure that the directory containing the HDFS `core-site.xml` file is in `gg.classpath`. This is so the `core-site.xml` file can be read at runtime and the connectivity information to HDFS can be resolved. For example:

```
gg.classpath={HDFS_install_directory}/etc/hadoop
```

If Kerberos authentication is enabled on the HDFS cluster, you have to configure the Kerberos principal and the location of the keytab file so that the password can be resolved at runtime:

```
gg.eventhandler.name.kerberosPrincipal=principal
gg.eventhandler.name.kerberosKeytabFile=path_to_the_keytab_file
```

10.2.21.3.2.2 About the Upstream Data Format

The Parquet Event Handler can only convert Avro Object Container File (OCF) generated by the File Writer Handler. The Parquet Event Handler cannot convert other formats to Parquet data files. The format of the File Writer Handler must be `avro_row_ocf` or `avro_op_ocf`, see [Flat Files](#).

10.2.21.3.3 Configuring the Parquet Event Handler

You configure the Parquet Event Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

The Parquet Event Handler works only in conjunction with the File Writer Handler.

To enable the selection of the Parquet Event Handler, you must first configure the handler type by specifying `gg.eventhandler.name.type=parquet` and the other Parquet Event properties as follows:

Table 10-25 Parquet Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.type</code>	Required	<code>parquet</code>	None	Selects the Parquet Event Handler for use.
<code>gg.eventhandler.name.writeToHDFS</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	Set to <code>false</code> to write to the local file system. Set to <code>true</code> to write directly to HDFS.

Table 10-25 (Cont.) Parquet Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.pathMappingTemplate</code>	Required	A string with resolvable keywords and constants used to dynamically generate the path to write generated Parquet files.	None	Use keywords interlaced with constants to dynamically generate unique path names at runtime. Typically, path names follow the format, <code>/ogg/data/\${groupName}/\${fullyQualifiedTableName}</code> . See Template Keywords .
<code>gg.eventhandler.name.fileNameMappingTemplate</code>	Optional	A string with resolvable keywords and constants used to dynamically generate the Parquet file name at runtime	None	Sets the Parquet file name. If not set, the upstream file name is used. See Template Keywords .
<code>gg.eventhandler.name.compressionCodec</code>	Optional	GZIP LZ0 SNAPPY UNCOMPRESSED	UNCOMPRESSED	Sets the compression codec of the generated Parquet file.
<code>gg.eventhandler.name.finalizeAction</code>	Optional	none delete	none	Indicates what the Parquet Event Handler should do at the finalize action. none Leave the data file in place. delete Delete the data file (such as, if the data file has been converted to another format or loaded to a third party application).
<code>gg.eventhandler.name.dictionaryEncoding</code>	Optional	true false	The Parquet default.	Set to <code>true</code> to enable Parquet dictionary encoding.
<code>gg.eventhandler.name.validation</code>	Optional	true false	The Parquet default.	Set to <code>true</code> to enable Parquet validation.
<code>gg.eventhandler.name.dictionaryPageSize</code>	Optional	Integer	The Parquet default.	Sets the Parquet dictionary page size.

Table 10-25 (Cont.) Parquet Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.maxPaddingSize</code>	Optional	Integer	The Parquet default.	Sets the Parquet padding size.
<code>gg.eventhandler.name.pageSize</code>	Optional	Integer	The Parquet default.	Sets the Parquet page size.
<code>gg.eventhandler.name.rowGroupSize</code>	Optional	Integer	The Parquet default.	Sets the Parquet row group size.
<code>gg.eventhandler.name.kerberosPrincipal</code>	Optional	The Kerberos principal name.	None	Set to the Kerberos principal when writing directly to HDFS and Kerberos authentication is enabled.
<code>gg.eventhandler.name.kerberosKeytabFile</code>	Optional	The path to the Kerberos keytab file.	The Parquet default.	Set to the path to the Kerberos keytab file with writing directly to HDFS and Kerberos authentication is enabled.
<code>gg.eventhandler.name.eventHandler</code>	Optional	A unique string identifier cross referencing a child event handler.	No event handler configured.	The event handler that is invoked on the file roll event. Event handlers can do file roll event actions like loading files to S3, converting to Parquet or ORC format, or loading files to HDFS.
<code>gg.eventhandler.name.writerVersion</code>	Optional	v1 v2	The Parquet library default which is up through Parquet version 1.11.0 is v1.	Allows the ability to set the Parquet writer version.

10.2.21.3.4 Parquet Event Handler Client Dependencies

What are the dependencies for the Parquet Event Handler?

The maven central repository artifacts for Parquet are:

Maven groupId: `org.apache.parquet`

Maven artifactId: `parquet-avro`

Maven version: `1.9.0`

Maven groupId: `org.apache.parquet`

Maven artifactId: `parquet-hadoop`

Maven version: 1.9.0

The Hadoop client dependencies are also required for the Parquet Event Handler, see [Hadoop Client Dependencies](#).

- [Parquet Client 1.12.0](#)
- [Parquet Client 1.11.1](#)
- [Parquet Client 1.10.1](#)
- [Parquet Client 1.9.0](#)

10.2.21.3.4.1 Parquet Client 1.12.0

audience-annotations-0.12.0.jar
avro-1.10.1.jar
commons-compress-1.20.jar
commons-pool-1.6.jar
jackson-annotations-2.11.3.jar
jackson-core-2.11.3.jar
jackson-databind-2.11.3.jar
javax.annotation-api-1.3.2.jar
parquet-avro-1.12.0.jar
parquet-column-1.12.0.jar
parquet-common-1.12.0.jar
parquet-encoding-1.12.0.jar
parquet-format-structures-1.12.0.jar
parquet-hadoop-1.12.0.jar
parquet-jackson-1.12.0.jar
slf4j-api-1.7.22.jar
snappy-java-1.1.8.jar
zstd-jni-1.4.9-1.jar

10.2.21.3.4.2 Parquet Client 1.11.1

audience-annotations-0.11.0.jar
avro-1.9.2.jar
commons-compress-1.19.jar
commons-pool-1.6.jar
jackson-annotations-2.10.2.jar
jackson-core-2.10.2.jar
jackson-databind-2.10.2.jar
javax.annotation-api-1.3.2.jar
parquet-avro-1.11.1.jar
parquet-column-1.11.1.jar
parquet-common-1.11.1.jar
parquet-encoding-1.11.1.jar
parquet-format-structures-1.11.1.jar
parquet-hadoop-1.11.1.jar
parquet-jackson-1.11.1.jar
slf4j-api-1.7.22.jar
snappy-java-1.1.7.3.jar

10.2.21.3.4.3 Parquet Client 1.10.1

avro-1.8.2.jar
commons-codec-1.10.jar
commons-compress-1.8.1.jar
commons-pool-1.6.jar
fastutil-7.0.13.jar
jackson-core-asl-1.9.13.jar
jackson-mapper-asl-1.9.13.jar

```
paranamer-2.7.jar  
parquet-avro-1.10.1.jar  
parquet-column-1.10.1.jar  
parquet-common-1.10.1.jar  
parquet-encoding-1.10.1.jar  
parquet-format-2.4.0.jar  
parquet-hadoop-1.10.1.jar  
parquet-jackson-1.10.1.jar  
slf4j-api-1.7.2.jar  
snappy-java-1.1.2.6.jar  
xz-1.5.jar
```

10.2.21.3.4.4 Parquet Client 1.9.0

```
avro-1.8.0.jar  
commons-codec-1.5.jar  
commons-compress-1.8.1.jar  
commons-pool-1.5.4.jar  
fastutil-6.5.7.jar  
jackson-core-asl-1.9.11.jar  
jackson-mapper-asl-1.9.11.jar  
paranamer-2.7.jar  
parquet-avro-1.9.0.jar  
parquet-column-1.9.0.jar  
parquet-common-1.9.0.jar  
parquet-encoding-1.9.0.jar  
parquet-format-2.3.1.jar  
parquet-hadoop-1.9.0.jar  
parquet-jackson-1.9.0.jar  
slf4j-api-1.7.7.jar  
snappy-java-1.1.1.6.jar  
xz-1.5.jar
```

10.2.22 Google BigQuery

Topics:

- [BigQuery Streaming Handler](#)
- [Google BigQuery Stage and Merge](#)

10.2.22.1 BigQuery Streaming Handler

- [BigQuery Streaming Handler with Legacy Streaming API](#)
Learn how to use the Google BigQuery Handler, which streams change data capture data from source trail files into Google BigQuery.
- [BigQuery Streaming API with Storage Write API](#)

10.2.22.1.1 BigQuery Streaming Handler with Legacy Streaming API

Learn how to use the Google BigQuery Handler, which streams change data capture data from source trail files into Google BigQuery.

BigQuery is a RESTful web service that enables interactive analysis of massively large datasets working in conjunction with Google Storage, see <https://cloud.google.com/bigquery/>.

- [Detailing the Functionality](#)

- [Setting Up and Running the BigQuery Handler](#)
The Google BigQuery Handler uses the Java BigQuery client libraries to connect to Big Query.
- [Google BigQuery Dependencies](#)
The Google BigQuery client libraries are required for integration with BigQuery.

10.2.22.1.1.1 Detailing the Functionality

- [Data Types](#)
- [Metadata Support](#)
- [Operation Modes](#)
- [Operation Processing Support](#)
- [Proxy Settings](#)
- [Mapping to Google Datasets](#)
A dataset is contained within a specific Google cloud project. Datasets are top-level containers that are used to organize and control access to your tables and views.

10.2.22.1.1.1.1 Data Types

The BigQuery Handler supports the standard SQL data types and most of these data types are supported by the BigQuery Handler. A data type conversion from the column value in the trail file to the corresponding Java type representing the BigQuery column type in the BigQuery Handler is required.

The following data types are supported:

BIGNUMERIC
BOOLEAN
BYTES
DATE
DATETIME
FLOAT
INTEGER
JSON
NUMERIC
STRING
TIME
TIMESTAMP

The BigQuery Handler does not support complex data types, such as ARRAY and STRUCT.

10.2.22.1.1.1.2 Metadata Support

The BigQuery Handler creates tables in BigQuery if the tables do not exist.

The BigQuery Handler alters tables to add columns which exist in the source metadata or configured metacolumns which do not exist in the target metadata. The BigQuery Handler also adds columns dynamically at runtime if it detects a metadata change.

The BigQuery Handler does not drop columns in the BigQuery table which do not exist into the source table definition. BigQuery neither supports dropping existing columns, nor supports changing the data type of existing columns. Once a column is created in BigQuery, it is immutable.

Truncate operations are not supported.

10.2.22.1.1.1.3 Operation Modes

You can configure the BigQuery Handler in one of these two modes:

Audit Log Mode = true

```
gg.handler.name.auditLogMode=true
```

When the handler is configured to run with audit log mode true, the data is pushed into Google BigQuery without a unique row identification key. As a result, Google BigQuery is not able to merge different operations on the same row. For example, a source row with an insert operation, two update operations, and then a delete operation would show up in BigQuery as four rows, one for each operation.

Also, the order in which the audit log is displayed in the BigQuery data set is not deterministic. To overcome these limitations, users should specify optype and position in the meta columns template for the handler. This adds two columns of the same names in the schema for the table in Google BigQuery. For example: `gg.handler.bigquery.metaColumnsTemplate = ${optype}, ${position}`

The optype is important to determine the operation type for the row in the audit log.

To view the audit log in order of the operations processed in the trail file, specify position which can be used in the ORDER BY clause while querying the table in Google BigQuery. For example:

```
SELECT * FROM [projectId:datasetId.tableId] ORDER BY position
```

auditLogMode = false

```
gg.handler.name.auditLogMode=false
```

When the handler is configured to run with audit log mode false, the data is pushed into Google BigQuery using a unique row identification key. The Google BigQuery is able to merge different operations for the same row. However, the behavior is complex. The Google BigQuery maintains a finite deduplication period in which it will merge operations for a given row. Therefore, the results can be somewhat non-deterministic.

The trail source needs to have a full image of the records in order to merge correctly.

Example 1

An insert operation is sent to BigQuery and before the deduplication period expires, an update operation for the same row is sent to BigQuery. The resultant is a single row in BigQuery for the update operation.

Example 2

An insert operation is sent to BigQuery and after the deduplication period expires, an update operation for the same row is sent to BigQuery. The resultant is that both the insert and the update operations show up in BigQuery.

This behavior has confounded many users, as this is the documented behavior when using the BigQuery SDK and a feature as opposed to a defect. The documented length of the deduplication period is at least one minute. However, Oracle testing has shown that the period is significantly longer. Therefore, unless users can guarantee that all operations for a give row occur within a very short period, it is likely there will be multiple entries for a given row in BigQuery. It is therefore just as important for users to configure meta columns with the optype and position so they can determine the latest state for a given row. To read more about audit log mode read the following Google BigQuery documentation: [Streaming data into BigQuery](#).

10.2.22.1.1.1.4 Operation Processing Support

The BigQuery Handler pushes operations to Google BigQuery using synchronous API. Insert, update, and delete operations are processed differently in BigQuery than in a traditional RDBMS.

The following explains how insert, update, and delete operations are interpreted by the handler depending on the mode of operation:

auditLogMode = true

- insert – Inserts the record with optype as an insert operation in the BigQuery table.
- update – Inserts the record with optype as an update operation in the BigQuery table.
- delete – Inserts the record with optype as a delete operation in the BigQuery table.
- pkUpdate—When pkUpdateHandling property is configured as delete-insert, the handler sends out a delete operation followed by an insert operation. Both these rows have the same position in the BigQuery table, which helps to identify it as a primary key operation and not a separate delete and insert operation.

auditLogMode = false

- insert – If the row does not already exist in Google BigQuery, then an insert operation is processed as an insert. If the row already exists in Google BigQuery, then an insert operation is processed as an update. The handler sets the deleted column to false.
- update – If a row does not exist in Google BigQuery, then an update operation is processed as an insert. If the row already exists in Google BigQuery, then an update operation is processed as update. The handler sets the deleted column to false.
- delete – If the row does not exist in Google BigQuery, then a delete operation is added. If the row exists in Google BigQuery, then a delete operation is processed as a delete. The handler sets the deleted column to true.
- pkUpdate—When pkUpdateHandling property is configured as delete-insert, the handler sets the deleted column to true for the row whose primary key is updated. It is followed by a separate insert operation with the new primary key and the deleted column set to false for this row.

Do not toggle the audit log mode because it forces the BigQuery handler to abend as Google BigQuery cannot alter schema of an existing table. The existing table needs to be deleted before switching audit log modes.

Note

The BigQuery Handler does not support the truncate operation. It abends when it encounters a truncate operation.

10.2.22.1.1.1.5 Proxy Settings

To connect to BigQuery using a proxy server, you must configure the proxy host and the proxy port in the properties file as follows:

```
jvm.bootoptions= -Dhttps.proxyHost=proxy_host_name -  
Dhttps.proxyPort=proxy_port_number
```

10.2.22.1.1.1.6 Mapping to Google Datasets

A dataset is contained within a specific Google cloud project. Datasets are top-level containers that are used to organize and control access to your tables and views.

A table or view must belong to a dataset, so you need to create at least one dataset before loading data into BigQuery.

The BigQuery handler can use existing datasets or create datasets if not found.

The BigQuery handler maps the table's schema name to the dataset name. For three-part table names, the dataset is constructed by concatenating catalog and schema.

10.2.22.1.1.2 Setting Up and Running the BigQuery Handler

The Google BigQuery Handler uses the Java BigQuery client libraries to connect to Big Query.

These client libraries are located using the following Maven coordinates:

- Group ID: `com.google.cloud`
- Artifact ID: `google-cloud-bigquery`
- Version: `2.7.1`

The BigQuery Client libraries do not ship with Oracle GoldenGate for Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA). Additionally, Google appears to have removed the link to download the BigQuery Client libraries. You can download the BigQuery Client libraries using Maven and the Maven coordinates listed above. However, this requires proficiency with Maven. The Google BigQuery client libraries can be downloaded using the Dependency downloading scripts. For more information, see [Google BigQuery Dependencies](#).

For more information about Dependency Downloader, see [Dependency Downloader](#).

- [Schema Mapping for BigQuery](#)
- [Understanding the BigQuery Handler Configuration](#)
- [Review a Sample Configuration](#)
- [Configuring Handler Authentication](#)

10.2.22.1.1.2.1 Schema Mapping for BigQuery

The table schema name specified in the replicat map statement is mapped to the BigQuery dataset name. For example: `map QASOURCE.*, target "dataset_US".*`;

This map statement replicates tables to the BigQuery dataset "dataset_US". Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) normalizes schema and table names to uppercase. Lowercase and mixed case dataset and table names are supported, but need to be quoted in the Replicat mapping statement.

10.2.22.1.1.2.2 Understanding the BigQuery Handler Configuration

The following are the configurable values for the BigQuery Handler. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the BigQuery Handler, you must first configure the handler type by specifying `gg.handler.name.type=bigquery` and the other BigQuery properties as follows:

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handlerlist</code>	Required	Any string	None	Provides a name for the BigQuery Handler. The BigQuery Handler name then becomes part of the property names listed in this table.
<code>gg.handler.name.type=bigquery</code>	Required	bigquery	None	Selects the BigQuery Handler for streaming change data capture into Google BigQuery.
<code>gg.handler.name.createDataset</code>	Optional	true false	true	Set to true to automatically create the BigQuery dataset if it does not exist.

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.credentialsFile</code>	Optional	Relative or absolute path to the credentials file	None	The credentials file downloaded from Google BigQuery for authentication. If you do not specify the path to the credentials file, you need to set it as an environment variable, see Configuring Handler Authentication .
<code>gg.handler.name.projectId</code>	Required	Any string	None	The name of the project in Google BigQuery. The handler needs project ID to connect to Google BigQuery store.
<code>gg.handler.name.batchSize</code>	Optional	Any number	500	The maximum number of operations to be batched together. This is applicable for all target table batches.
<code>gg.handler.name.batchFlushFrequency</code>	Optional	Any number	1000	The maximum amount of time in milliseconds to wait before executing the next batch of operations. This is applicable for all target table batches.
<code>gg.handler.name.skipInvalidRows</code>	Optional	true false	false	Sets whether to insert all valid rows of a request, even if invalid rows exist. If not set, the entire insert request fails if it contains an invalid row.
<code>gg.handler.name.ignoreUnknownValues</code>	Optional	true false	false	Sets whether to accept rows that contain values that do not match the schema. If not set, rows with unknown values are considered to be invalid.
<code>gg.handler.name.connectionTimeout</code>	Optional	Positive integer	20000	The maximum amount of time, in milliseconds, to wait for the handler to establish a connection with Google BigQuery.
<code>gg.handler.name.readTimeout</code>	Optional	Positive integer	30000	The maximum amount of time in milliseconds to wait for the handler to read data from an established connection.
<code>gg.handler.name.metaColumnsTemplate</code>	Optional	A legal string	None	A legal string specifying the metaColumns to be included. If you set <code>auditLogMode</code> to true, it is important that you set the <code>metaColumnsTemplate</code> property to view the operation type for the row inserted in the audit log, see Metacolumn Keywords .
<code>gg.handler.name.auditLogMode</code>	Optional	true false	false	Set to true, the handler writes each record to the target without any primary key. Everything is processed as insert. Set to false, the handler tries to merge incoming records into the target table if they have the same primary key. Primary keys are needed for this property. The trail source records need to have a full image updates to merge correctly.

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.pkUpdateHandling</code>	Optional	<code>abend</code> <code>delete-insert</code>	<code>abend</code>	Sets how the handler handles update operations that change a primary key. Primary key operations can be problematic for the BigQuery Handler and require special consideration: <code>abend</code>- indicates the process abends. <code>delete-insert</code>- indicates the process treats the operation as a delete and an insert. The full before image is required for this property to work correctly. Without full before and after row images the insert data are incomplete. Oracle recommends this option.
<code>gg.handler.name.adjustScale</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	The BigQuery numeric data type supports a maximum scale of 9 digits. If a field is mapped into a BigQuery numeric data type, then it fails if the scale is larger than 9 digits. Set this property to <code>true</code> to round fields mapped to BigQuery numeric data types to a scale of 9 digits. Enabling this property results in a loss of precision for source data values with a scale larger than 9.
<code>gg.handler.name.includeDeletedColumn</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	Set to <code>true</code> to include a boolean column in the output called <code>deleted</code> . The value of this column is set to <code>false</code> for insert and update operations, and is set to <code>true</code> for delete operations.
<code>gg.handler.name.enableAlter</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	Set to <code>true</code> to enable altering the target BigQuery table. This will allow the BigQuery Handler to add columns or metacolumns configured on the source, which are not currently in the target BigQuery table.
<code>gg.handler.name.clientId</code>	Optional	String	None	Use to set the client id if the configuration property <code>gg.handler.name.credentialsFile</code> to resolve the Google BigQuery credentials is not set. You may wish to use this property instead of the credentials file in order to use Oracle Wallet to secure credentials.
<code>gg.handler.name.clientEmail</code>	Optional	String	None	Use to set the client email if the configuration property <code>gg.handler.name.credentialsFile</code> to resolve the Google BigQuery credentials is not set. You may wish to use this property instead of the credentials file in order to use Oracle Wallet to secure credentials.
<code>gg.handler.name.privateKey</code>	Optional	String	None	Use to set the private key if the configuration property <code>gg.handler.name.credentialsFile</code> to resolve the Google BigQuery credentials is not set. You may wish to use this property instead of the credentials file in order to use Oracle Wallet to secure credentials.

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.privateKeyId</code>	Optional	String	None	Use to set the private key id if the configuration property <code>gg.handler.name.credentialsFile</code> to resolve the Google BigQuery credentials is not set. You may wish use this property instead of the credentials file in order to use Oracle Wallet to secure credentials.
<code>gg.handler.name.url</code>	Optional	A legal URL to connect to BigQuery including scheme, server name and port (if not the default port). The default is <code>https://www.googleapis.com</code> .	<code>https://www.googleapis.com</code>	Allows the user to set a URL for a private endpoint to connect to BigQuery.

To be able to connect GCS to the Google Cloud Service account, ensure that either of the following is configured: the credentials file property with the relative or absolute path to credentials JSON file or the properties for individual credentials keys. The configuration property that is used to individually add google service account credential key enables them to be encrypted using the Oracle wallet.

10.2.22.1.1.2.3 Review a Sample Configuration

The following is a sample configuration for the BigQuery Handler:

```
gg.handlerlist = bigquery

#The handler properties
gg.handler.bigquery.type = bigquery
gg.handler.bigquery.projectId = festive-athlete-201315
gg.handler.bigquery.credentialsFile = credentials.json
gg.handler.bigquery.auditLogMode = true
gg.handler.bigquery.pkUpdateHandling = delete-insert

gg.handler.bigquery.metaColumnsTemplate = ${optype}, ${position}
```

10.2.22.1.1.2.4 Configuring Handler Authentication

You have to configure the BigQuery Handler authentication using the credentials in the JSON file downloaded from Google BigQuery.

Download the credentials file:

1. Login into your Google account at cloud.google.com.
2. Click **Console**, and then go to the Dashboard where you can select your project.
3. From the navigation menu, click **APIs & Services** then select **Credentials**.
4. From the Create Credentials menu, choose **Service account key**.
5. Choose the JSON key type to download the JSON credentials file for your system.

After you have the credentials file, you can authenticate the handler in one of the following methods listed here:

- Specify the path to the credentials file in the properties file with the `gg.handler.name.credentialsFile` configuration property.

The path of the credentials file must contain the path with no wildcard appended. If you include the * wildcard in the path to the credentials file, the file is not recognized.

Or

- Set the credentials file keys (`clientId`, `clientEmail`, `privateKeyId`, and `privateKey`) into the corresponding handler properties.

Or

- Set the `GOOGLE_APPLICATION_CREDENTIALS` environment variable on your system. For example:

```
export GOOGLE_APPLICATION_CREDENTIALS = credentials.json
```

Then restart the Oracle GoldenGate manager process.

10.2.22.1.1.3 Google BigQuery Dependencies

The Google BigQuery client libraries are required for integration with BigQuery.

The maven coordinates are as follows:

Maven groupId: `com.google.cloud`

Maven artifactId: `google-cloud-bigquery`

Version: `2.7.1`

- [BigQuery 2.7.1](#)

10.2.22.1.1.3.1 BigQuery 2.7.1

The required BigQuery Client libraries for the 2.7.1 version are as follows:

```
api-common-2.1.3.jar
checker-compat-qual-2.5.5.jar
checker-qual-3.21.1.jar
commons-codec-1.15.jar
commons-logging-1.2.jar
error_prone_annotations-2.11.0.jar
failureaccess-1.0.1.jar
gax-2.11.0.jar
gax-httpjson-0.96.0.jar
google-api-client-1.33.1.jar
google-api-services-bigquery-v2-rev20211129-1.32.1.jar
google-auth-library-credentials-1.4.0.jar
google-auth-library-oauth2-http-1.4.0.jar
```

```
google-cloud-bigquery-2.7.1.jar
google-cloud-core-2.4.0.jar
google-cloud-core-http-2.4.0.jar
google-http-client-1.41.2.jar
google-http-client-apache-v2-1.41.2.jar
google-http-client-appengine-1.41.2.jar
google-http-client-gson-1.41.2.jar
google-http-client-jackson2-1.41.2.jar
google-oauth-client-1.33.0.jar
grpc-context-1.44.0.jar
gson-2.8.9.jar
guava-31.0.1-jre.jar
httpclient-4.5.13.jar
httpcore-4.4.15.jar
j2objc-annotations-1.3.jar
jackson-core-2.13.1.jar
javax.annotation-api-1.3.2.jar
jsr305-3.0.2.jar
listenablefuture-9999.0-empty-to-avoid-conflict-with-guava.jar
opencensus-api-0.31.0.jar
opencensus-contrib-http-util-0.31.0.jar
protobuf-java-3.19.3.jar
protobuf-java-util-3.19.3.jar
proto-google-common-protos-2.7.2.jar
proto-google-iam-v1-1.2.1.jar
```

10.2.22.1.2 BigQuery Streaming API with Storage Write API

Overview

BigQuery is a fully managed, AI-ready data platform, hosted on Google Cloud Platform (GCP). This handler uses recommended Storage Write API for low latency streaming of insert records without using any staging.

- [Detailed Functionality](#)
- [Database User Privileges](#)
- [Prerequisites](#)
- [Configuration](#)
- [Classpath Configuration](#)
- [Proxy Configuration](#)
- [Sample Configuration](#)
- [Troubleshooting and Diagnostics](#)

10.2.22.1.2.1 Detailed Functionality

The Storage Write API of Google BigQuery has many advantages over legacy API and it is the recommended technology for streaming insert only records to target without any staging at low latency. It also ensures exactly once apply ensuring no duplicates or missing records. For more information, see <https://cloud.google.com/java/docs/reference/google-cloud-bigquerystorage/latest/overview>

Note

Oracle GoldenGate uses the Storage Write API in buffered type which guarantees the exactly-once semantics. For more information, see [Google BigQuery Storage Write API](#).

10.2.22.1.2.2 Database User Privileges

The service account used for replicating insert records into Big Query must be granted either of IAM roles `bigquery.dataEditor`, `bigquery.dataOwner`, or `bigquery.admin` for inserting records to target and automatic creation of table. For more information, see <https://cloud.google.com/bigquery/docs/access-control>

10.2.22.1.2.3 Prerequisites

- Oracle GoldenGate trails must be configured to generate INSERT operations only.
- Oracle GoldenGate must be configured with auto-creation of table enabled if the target table does not exist on target.
- If you want to maintain the source operation types in the target Google BigQuery tables while using BigQuery Streaming API with Storage Write API, use the COLMAP type. For example:

```
"MAP QASOURCE.TCUSTORD, TARGET QASOURCE.TCUSTORD, COLMAP (USEDEFAULTS,
op_type = @GETENV('GGHEADER','OPTYPE'));
```

- If you do not want to maintain the source operation types in the target Google BigQuery tables while using BigQuery Streaming API with Storage Write API, all the source operations will be ingested as the INSERT type to target Google BigQuery tables.

10.2.22.1.2.4 Configuration

The configuration of BigQuery replication properties is stored in Replicat properties file.

Note

Ensure to specify the path to the properties file in the parameter file only when using Coordinated Replicat. Add the following line to the parameter file: `TARGETDB LIBFILE libggjava.so SET property=<parameter file directory>/<properties file name>`

The following are configuration properties available for the BigQuery Streaming handler, the required ones must be changed to match your BigQuery Streaming configuration:

Property Name	Required	Property Value	Default	Description
gg.handler.hand ler-name.type	Yes	[bigquerystream ing]	N.A.	Defines use of Big Query Streaming replicat

Property Name	Required	Property Value	Default	Description
gg.handler.handler-name.url	No	[private endpoint URL]	[public endpoint URL is added by default]	Defines the endpoint URL. For private endpoint, URL must be configured.
gg.handler.handler-name.credentialsFile	No	[Credentials JSON file path for GCP service account]	N.A.	Path for GCP credentials JSON file for service account for securely connecting to GCP.
gg.handler.handler-name.clientId	No	[Valid client Id]	N.A.	Individual credentials parameter for GCP service account.
gg.handler.handler-name.clientEmail	No	Valid service account email [<service-account-name>@<projectId>.iam.gserviceaccount.com]	N.A.	Individual credentials parameter for GCP service account.
gg.handler.handler-name.privateKeyId	No	[Valid service account privateKeyId]	N.A.	Individual credentials parameter for GCP service account.
gg.handler.handler-name.privateKey	No	[Valid service account private key]	N.A.	Individual credentials parameter for GCP service account.
gg.handler.handler-name.projectId	No	[Valid BigQuery Project Id]	N.A.	Big Query Project Id used for replication.
gg.handler.handler-name.maxRetryDelayDuration	No	[duration in sec.]	60	Maximum duration for stream writer to retry data write (append).
gg.handler.handler-name.maxRetryAttempt	No	Number of attempts	5	Maximum number for retry attempted by stream writer before failing to write.
gg.handler.handler-name.StreamBatchSize	No	[number of records]	[5000]	Max record that can be buffered before flushing.
gg.handler.handler-name.createTable	No	[true/false]	True	Auto-creation of table and dataset on target is enabled or not. This will be auto configured to enabled.
gg.handler.handler-name.createDataset	No	[true/false]	True	Auto-creation of dataset enabling.

Property Name	Required	Property Value	Default	Description
gg.handler.handler-name.metaColumnsTemplate	No	[valid template string]	N.A.	Defines meta column template for the table.

10.2.22.1.2.5 Classpath Configuration

Google BigQuery Streaming Handler uses the Java SDK for Big Query and Storage API.

Ensure that the gg.classpath configuration parameter includes the path to the ingest SDK.

- [Maven Coordinates](#)
- [Dependencies](#)

10.2.22.1.2.5.1 Maven Coordinates

The BigQuery Streaming relies on BigQuery Storage API Java SDK, the SDK jar needs to be downloaded to the location provided in the class path.

The maven coordinates for the dependency jars of SDK are:

```
<dependency>
  <groupId>com.google.cloud</groupId>
  <artifactId>google-cloud-bigquerystorage</artifactId>
  <version>3.9.2</version>
</dependency>
<dependency>
  <groupId>com.google.cloud</groupId>
  <artifactId>google-cloud-bigquery</artifactId>
  <version>2.43.0</version>
</dependency>
```

10.2.22.1.2.5.2 Dependencies

You can download using the Dependency Downloader script to download the dependencies by running the following script:

```
<OGGDIR>/DependencyDownloader/bigquerystreaming.sh
```

For more information about Dependency Downloader, see [Dependency Downloader](#).

10.2.22.1.2.6 Proxy Configuration

When the Replicat process runs behind a proxy server, the property jvm.bootoptions can be used to set proxy server configuration.

For example: jvm.bootoptions=-Dhttps.proxyHost=<some-proxy-address.com> -Dhttps.proxyPort=80

10.2.22.1.2.7 Sample Configuration

The sample properties file can also be found in the directory <OGGDIR>/AdapterExamples/big-data/bigquery_streaming/.

Note: Recommended to only edit the configuration marked as TODO

```
gg.handlerlist=bqs
gg.handler.bqs.type=bigquerystreaming
#TODO: Set the credentials file with service account info for connection
gg.handler.bqs.credentialsFile=/path/to/creds.json
```

```
#TODO: Set the classpath to include Big Query storage SDK  
gg.classpath=./bigquery-streaming-dep/*
```

10.2.22.1.2.8 Troubleshooting and Diagnostics

- **Connectivity issues to BigQuery:**
 - Validate service account configuration parameters: `project_id`, `client_id`, `client_email`, `private_key`, or `private_key_id` etc.
 - Check HTTP(S) proxy configuration if running Replicat process behind a proxy.
- **DDL not applied on the target table:** Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) does not support DDL replication.
- **Stream Errors:** In case there are any errors while ingesting records with Stream, the Stream info along with the associated table name is logged into the GoldenGate for Big Data handler log file.

10.2.22.2 Google BigQuery Stage and Merge

Topics:

- [Overview](#)
BigQuery is Google Cloud's fully managed, petabyte-scale, and cost-effective analytics data warehouse that lets you run analytics over vast amounts of data in near real time.
- [Detailed Functionality](#)
- [Prerequisites](#)
- [Differences between BigQuery Handler and Stage and Merge BigQuery Event Handler](#)
- [Authentication or Authorization](#)
- [Operation Aggregation](#)
- [Compressed Update Handling](#)
- [Configuration](#)
- [Troubleshooting and Diagnostics](#)
- [Restrictions on Database Object Names](#)

10.2.22.2.1 Overview

BigQuery is Google Cloud's fully managed, petabyte-scale, and cost-effective analytics data warehouse that lets you run analytics over vast amounts of data in near real time.

10.2.22.2.2 Detailed Functionality

The BigQuery Event handler uses the stage and merge data flow.

The change data is staged in a temporary location in microbatches and eventually merged into the target table. Google Cloud Storage (GCS) is used as the staging area for change data.

This Event handler is used as a downstream Event handler connected to the output of the GCS Event handler.

The GCS Event handler loads files generated by the File Writer Handler into Google Cloud Storage.

The Event handler runs BigQuery Query jobs to execute MERGE SQL. The SQL operations are performed in batches providing better throughput.

Note

The BigQuery Event handler doesn't use the Google BigQuery streaming API.

10.2.22.2.3 Prerequisites

Google Cloud Storage (GCS) bucket and dataset location: Ensure that the GCS bucket and the BigQuery dataset exist in the same location/region.

10.2.22.2.4 Differences between BigQuery Handler and Stage and Merge BigQuery Event Handler

Table 10-26 BigQuery Handler v/s Stage and Merge BigQuery Event Handler

Feature/Limitation	BigQuery Handler	Stage And Merge BigQuery Event Handler
Compressed update support	Partially supported with limitations.	YES
Audit log mode	Process all the operations as INSERT.	No need to enable audit log mode.
GCP Quotas/Limits	Maximum rows per second per table: 100000. See Google BigQuery Documentation .	Daily destination table update limit — 1500 updates per table per day. See Google BigQuery Documentation .
Approximate pricing with 1TB Storage (for exact pricing refer GCP Pricing calculator)	Streaming Inserts for 1TB costs ~72.71 USD per month	Query job for 1TB costs ~20.28 USD per month.
Duplicate rows replicated to BigQuery	YES	NO
Replication of TRUNCATE operation	Not supported	Supported
API used	BigQuery Streaming API	BigQuery Query job

10.2.22.2.5 Authentication or Authorization

For more information about using the Google service account key, see [Authentication and Authorization](#) in the Google Cloud Service (GCS) Event Handler topic. In addition to the permissions needed to access GCS, the service account also needs permissions to access BigQuery. You may choose to use a pre-defined IAM role, such as roles/bigquery.dataEditor or roles/bigquery.dataOwner. When creating a custom role, the following are the IAM permissions used to run BigQuery Event handler. For more information, see [Configuring Handler Authentication](#).

- [BigQuery Permissions](#)

10.2.22.2.5.1 BigQuery Permissions

Table 10-27 BigQuery Permissions

Permission	Description
<code>bigquery.connections.create</code>	Create new connections in a project.
<code>bigquery.connections.delete</code>	Delete a connection.
<code>bigquery.connections.get</code>	Gets connection metadata. Credentials are excluded.
<code>bigquery.connections.list</code>	List connections in a project.
<code>bigquery.connections.update</code>	Update a connection and its credentials.
<code>bigquery.connections.use</code>	Use a connection configuration to connect to a remote data source.
<code>bigquery.datasets.create</code>	Create new datasets.
<code>bigquery.datasets.get</code>	Get metadata about a dataset.
<code>bigquery.connections.export</code>	Export table data out of BigQuery.
<code>bigquery.connections.get</code>	Get table metadata. To get table data, you need <code>bigquery.tables.getData</code> .
<code>bigquery.connections.list</code>	List connections in a project.
<code>bigquery.connections.update</code>	Update a connection and its credentials.
<code>bigquery.datasets.create</code>	Create new empty datasets.
<code>bigquery.datasets.get</code>	Get metadata about a dataset.
<code>bigquery.datasets.getIamPolicy</code>	Reserved for future use.
<code>bigquery.datasets.update</code>	Update metadata for a dataset.
<code>bigquery.datasets.updateTag</code>	Update tags for a dataset.
<code>bigquery.jobs.create</code>	Run jobs (including queries) within the project.
<code>bigquery.jobs.get</code>	Get data and metadata on any job.
<code>bigquery.jobs.list</code>	List all jobs and retrieve metadata on any job submitted by any user. For jobs submitted by other users, details and metadata are redacted.
<code>bigquery.jobs.listAll</code>	List all jobs and retrieve metadata on any job submitted by any user.
<code>bigquery.jobs.update</code>	Cancel any job.
<code>bigquery.readsessions.create</code>	Create a new read session via the BigQuery Storage API.
<code>bigquery.readsessions.getData</code>	Read data from a read session via the BigQuery Storage API.
<code>bigquery.readsessions.update</code>	Update a read session via the BigQuery Storage API.
<code>bigquery.reservations.create</code>	Create a reservation in a project.
<code>bigquery.reservations.delete</code>	Delete a reservation.
<code>bigquery.reservations.get</code>	Retrieve details about a reservation.
<code>bigquery.reservations.list</code>	List all reservations in a project.
<code>bigquery.reservations.update</code>	Update a reservation's properties.

Table 10-27 (Cont.) BigQuery Permissions

Permission	Description
<code>bigquery.reservationAssignments.create</code>	Create a reservation assignment. This permission is required on the owner project and assignee resource. To move a reservation assignment, you need <code>bigquery.reservationAssignments.create</code> on the new owner project and assignee resource.
<code>bigquery.reservationAssignments.delete</code>	Delete a reservation assignment. This permission is required on the owner project and assignee resource. To move a reservation assignment, you need <code>bigquery.reservationAssignments.delete</code> on the old owner project and assignee resource.
<code>bigquery.reservationAssignments.list</code>	List all reservation assignments in a project.
<code>bigquery.reservationAssignments.search</code>	Search for a reservation assignment for a given project, folder, or organization.
<code>bigquery.routines.create</code>	Create new routines (functions and stored procedures).
<code>bigquery.routines.delete</code>	Delete routines.
<code>bigquery.routines.list</code>	List routines and metadata on routines.
<code>bigquery.routines.update</code>	Update routine definitions and metadata.
<code>bigquery.savedqueries.create</code>	Create saved queries.
<code>bigquery.savedqueries.delete</code>	Delete saved queries.
<code>bigquery.savedqueries.get</code>	Get metadata on saved queries.
<code>bigquery.savedqueries.list</code>	Lists saved queries.
<code>bigquery.savedqueries.update</code>	Updates saved queries.
<code>bigquery.tables.create</code>	Create new tables.
<code>bigquery.tables.delete</code>	Delete tables
<code>bigquery.tables.export</code>	Export table data out of BigQuery.
<code>bigquery.tables.get</code>	Get table metadata. To get table data, you need <code>bigquery.tables.getData</code> .
<code>bigquery.tables.getData</code>	Get table data. This permission is required for querying table data. To get table metadata, you need <code>bigquery.tables.get</code> .
<code>bigquery.tables.getIamPolicy</code>	Read a table's IAM policy.
<code>bigquery.tables.list</code>	List tables and metadata on tables.
<code>bigquery.tables.setCategory</code>	Set policy tags in table schema.
<code>bigquery.tables.setIamPolicy</code>	Changes a table's IAM policy.
<code>bigquery.tables.update</code>	Update table metadata. To update table data, you need <code>bigquery.tables.updateData</code> .
<code>bigquery.tables.updateData</code>	Update table data. To update table metadata, you need <code>bigquery.tables.update</code> .
<code>bigquery.tables.updateTag</code>	Update tags for a table.

In addition to these permissions, ensure that `resourcemanager.projects.get/list` is always granted as a pair.

10.2.22.2.6 Operation Aggregation

Operation aggregation is the process of aggregating (merging/compressing) multiple operations on the same row into a single output operation based on a threshold.

- [In-Memory Operation Aggregation](#)
- [Operation Aggregation Using SQL](#)

10.2.22.2.6.1 In-Memory Operation Aggregation

- Operation records are aggregated in-memory by default.
- The `gg.aggregate.operations.flush.interval` property has been deprecated and is no longer supported. If `gg.aggregate.operations.flush.interval` is used in GG for DAA 23ai, then replicat will run; but add a warning to log file about the property being deprecated and not supported.
To control the time window for aggregation, use the `gg.handler.bq.fileRollInterval` property. By default, it is set to 3 minutes. Longer intervals will increase latency, and may increase memory usage. Shorter intervals will increase overhead in Oracle GoldenGate and the target database.
- Operation aggregation in-memory requires additional JVM memory configuration.

10.2.22.2.6.2 Operation Aggregation Using SQL

- To use SQL aggregation, it is mandatory that the trail files contain uncompressed UPDATE operation records, which means that the UPDATE operations contain full image of the row being updated.
- Operation aggregation using SQL can provide better throughput if the trails files contains uncompressed update records.
- Replicat can aggregate operations using SQL statements by setting the `gg.aggregate.operations.using.sql=true`.
- You can tune the frequency of merge interval using the File writer `gg.handler.bq.fileRollInterval` property, the default value is set to 3m (three minutes).
- Operation aggregation using SQL does not require additional JVM memory configuration.

10.2.22.2.7 Compressed Update Handling

A compressed update record contains values for the key columns and the modified columns.

An uncompressed update record contains values for all the columns.

Oracle GoldenGate trails may contain compressed or uncompressed update records. The default extract configuration writes compressed updates to the trails. The parameter `gg.compressed.update` can be set to `true` or `false` to indicate compressed or uncompressed update records.

The default extract configuration writes compressed updates to the trails. The parameter `gg.compressed.update` can be set to `true` or `false` to indicate compressed/uncompressed update records.

- [MERGE Statement with Uncompressed Updates](#)

10.2.22.2.7.1 MERGE Statement with Uncompressed Updates

In some use cases, if the trail contains uncompressed update records, then the MERGE SQL statement can be optimized for better performance by setting `gg.compressed.update=false`.

If you want to use DELETE+INSERT SQL statements instead of a MERGE SQL statement, then set `gg.eventhandler.bq.deleteInsert=true`.

10.2.22.2.8 Configuration

- [Automatic Configuration](#)
- [Classpath Configuration](#)
The GCS Event handler and the BigQuery Event handler use the Java SDK provided by Google. Google does not provide a direct link to download the SDK.
- [Proxy Configuration](#)
- [INSERTALLRECORDS Support](#)
- [BigQuery Dataset and GCP ProjectId Mapping](#)
- [BigQuery Iceberg Tables](#)
- [End-to-End Configuration](#)

10.2.22.2.8.1 Automatic Configuration

Replication to BigQuery involves configuring of multiple components, such as File Writer handler, Google Cloud Storage (GCS) Event handler and BigQuery Event handler.

The Automatic Configuration functionality helps to auto configure these components so that the user configuration is minimal.

The properties modified by auto configuration is also logged in the handler log file. To enable auto configuration to replicate to BigQuery target, set the parameter `gg.target=bq`.

When replicating to BigQuery target, you cannot customize GCS Event handler name and BigQuery Event handler name.

- [File Writer Handler Configuration](#)
File Writer handler name is preset to the value `bq`. The following is an example to edit a property of File Writer handler: `gg.handler.bq.pathMappingTemplate=./dirout`.
- [GCS Event Handler Configuration](#)
The GCS Event handler name is preset to the value `gcs`. The following is an example to edit a property of GCS Event handler: `gg.eventhandler.gcs.concurrency=5`.
- [BigQuery Event Handler Configuration](#)
BigQuery Event handler name is preset to the value `bq`. There are no mandatory parameters required for BigQuery Event handler. Mostly, auto configure derives the required parameters.

10.2.22.2.8.1.1 File Writer Handler Configuration

File Writer handler name is preset to the value `bq`. The following is an example to edit a property of File Writer handler: `gg.handler.bq.pathMappingTemplate=./dirout`.

10.2.22.2.8.1.2 GCS Event Handler Configuration

The GCS Event handler name is preset to the value `gcs`. The following is an example to edit a property of GCS Event handler: `gg.eventhandler.gcs.concurrency=5`.

10.2.22.2.8.1.3 BigQuery Event Handler Configuration

BigQuery Event handler name is preset to the value bq. There are no mandatory parameters required for BigQuery Event handler. Mostly, auto configure derives the required parameters.

The following are the BigQuery Event handler configurations:

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler .bq.credentials File	Optional	Relative or absolute path to the service account key file.	Value from property gg.eventhandler .gcs.credentials sFile	Sets the path to the service account key file. Autoconfigure will automatically configure this property based on the configuration gg.eventhandler .gcs.credentials sFile, unless the user wants to use a different service account key file for BigQuery access. Alternatively, if the environment variable GOOGLE_APPLICATION_CREDENTIALS is set to the path to the service account key file, this parameter need not be set.
gg.eventhandler .bq.projectId	Optional	The Google project-id	project-id associated with the service account.	Sets the project-id of the Google Cloud project that houses BigQuery. Autoconfigure will automatically configure this property by accessing the service account key file unless user wants to override this explicitly.

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler .bq.kmsKey	Optional	Key names in the format: projects/<PROJECT>/locations/<LOCATION>/keyRings/<RING_NAME>/cryptoKeys/<KEY_NAME> • <PROJECT>: Google project-id • <LOCATION>: Location of the BigQuery dataset. • <RING_NAME>: Google Cloud KMS key ring name. • <KEY_NAME>: Google Cloud KMS key name.	Value from property gg.eventhandler.gcs.kmsKey	Set a customer managed Cloud KMS key to encrypt data in BigQuery. Autoconfigure will automatically configure this property based on the configuration gg.eventhandler.gcs.kmsKey.
gg.eventhandler .bq.connectionTimeout	Optional	Positive integer.	20000	The maximum amount of time, in milliseconds, to wait for the handler to establish a connection with Google BigQuery.
gg.eventhandler .bq.readTimeout	Optional	Positive integer.	30000	The maximum amount of time in milliseconds to wait for the handler to read data from an established connection.
gg.eventhandler .bq.totalTimeout	Optional	Positive integer.	120000	The total timeout parameter in seconds. The TotalTimeout parameter has the ultimate control over how long the logic should keep trying the remote call until it gives up completely.
gg.eventhandler .bq.createDataset	Optional	true false	true	Set to true to automatically create the BigQuery dataset if it does not exist.

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler .bq.createTable	Optional	true false	true	Set to true to automatically create the BigQuery target table if it does not exist.

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler.bq.fileRollInterval	Optional	Integer	The default unit of measure is milliseconds. You can stipulate ms, s, m, h to signify milliseconds, seconds, minutes, or hours respectively. Examples of legal values include 10000, 10000ms, 10s, 10m, or 1.5h. Values of 0 or less indicate that file rolling on time is turned off.	The parameter determines how often the data will be merged into Google BigQuery. Use with caution, the higher this value is the more data will need to be stored in the memory of the Replicat process.

Note
Use the parameter to determine how often the data will be merged into Google BigQuery. Use with caution, the higher this value is the more data will need to be stored in the memory of the Replicat process.

Properties	Required/ Optional	Legal Values	Default	Explanation

i
t
s
d
e
f
a
u
l
t
v
a
l
u
e
(
3
m
)
w
i
l
l
i
n
c
r
e
a
s
e
t
h
e
a
m
o
u
n
t
o
f
d
a
t
a
s
t
o
r
e
d
i
n
t

Properties	Required/ Optional	Legal Values	Default	Explanation

h
e
i
n
t
e
r
n
a
l
m
e
m
o
r
y
o
f
t
h
e
R
e
p
l
i
c
a
t
.
T
h
i
s
c
a
n
c
a
u
s
e
o
u
t
o
f
m
e
m
o
r
y
e
r

Properties	Required/ Optional	Legal Values	Default	Explanation

r
o
r
s
a
n
d
s
t
o
p
t
h
e
R
e
p
l
i
c
a
t
i
f
i
t
r
u
n
s
o
u
t
o
f
m
e
m
o
r
y
.

① N
o
t
e
S
t
a
r
t

ing with the 23 airlines, the average gate operation is fully

Properties	Required/ Optional	Legal Values	Default	Explanation

.
i
n
t
e
r
v
a
l
p
r
o
p
e
r
t
y
i
s
d
e
p
r
e
c
a
t
e
d
a
n
d
n
o
l
o
n
g
e
r
s
u
p
p
o
r
t
e
d
.
F
o
r
m
o

Properties	Required/ Optional	Legal Values	Default	Explanation

re information, see [Oracle GoldenGate Administrator's Guide](#)

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.compressed.update	Optional	true or false	true	If set the true, then this indicates that the source trail files contain compressed update operations. If set to true, then the source trail files are expected to contain uncompressed update operations.
gg.eventhandler.bq.connectionRetryIntervalSeconds	Optional	Integer Value	30	Specifies the delay in seconds between connection retry attempts.
gg.eventhandler.bq.connectionRetries	Optional	Integer Value	3	Specifies the number of times connections to the target data warehouse will be retried.
gg.eventhandler.bq.url	Optional	An absolute URL to connect to Google BigQuery.	https://googleapis.com	A legal URL to connect to Google BigQuery including scheme, server name and port (if not the default port). The default is https://googleapis.com.
gg.eventhandler.bq.clientId	Optional	Client Id param of the service account key file.	Value from property gg.eventhandler.gcs.clientId	Sets the clientId of the service account key file. Auto-configure will automatically configure this property based on the configuration gg.eventhandler.gcs.clientId, unless the user wants to use a different service account key file for BigQuery access. This enables service account key for encryption using Oracle wallet.

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler .bq.clientEmail	Optional	Client Email of the service account key file.	Value from property gg.eventhandler.gcs.clientEmail	Sets the client Email of the service account key file. Auto configure will automatically configure this property based on the configuration q, unless the user wants to use a different service account key file for BigQuery access. This enables service account key for encryption using Oracle wallet.
gg.eventhandler .bq.privateKeyId	Optional	Private Key ID of the service account key file.	Value from property gg.eventhandler.gcs.privateKeyId	Sets the private key Id of the service account key file. Auto-configure will automatically configure this property based on the configuration gg.eventhandler.gcs.privateKeyId, unless the user wants to use a different service account key file for BigQuery access. This enables service account key for encryption using Oracle wallet.
gg.eventhandler .bq.privateKey	Optional	Private Key of the service account key file.	Value from property gg.eventhandler.gcs.privateKey	Sets the private key of the service account key file. Auto configure will automatically configure this property based on the configuration gg.eventhandler.gcs.privateKey, unless the user wants to use a different service account key file for BigQuery access. This enables service account key for encryption using Oracle wallet.

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.validate.key update	Optional	true or false	false	If set to true, Replicat will validate key update operations (optype 115) and correct to normal update if no key values have changed. Compressed key update operations do not qualify for merge.
gg.eventhandler .bq.deleteInsert	Optional	true or false	false	If set to true, Replicat will merge records using SQL DELETE+INSERT statements instead of SQL MERGE statement. Applicable only if gg.compressed.update is set to false.
gg.eventhandler .bq.tableType	Optional	native or iceberg. For more information, see BigQuery Iceberg Tables .	native	Indicates BigQuery table format for automatic table creation. Options are native (default): automatically create native BigQuery tables, and iceberg: automatically create Iceberg tables. For more information, see BigQuery Iceberg Tables .
gg.eventhandler .bq.storageUri	Required	BigQuery Iceberg table's Cloud Storage URI.	None	Required when the property tableType is set to iceberg. A fully qualified Cloud Storage URI for Iceberg tables. For more information, see BigQuery Iceberg Tables .

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler .bq.storageConn ectionId	Required	Storage Connection ID to access the Bigquery Iceberg table's Cloud Storage URI.	None	Required when the property tableType is set to iceberg. Storage Connection ID to access the Cloud Storage URI for Iceberg tables. For more information, see BigQuery Iceberg Tables .

10.2.22.2.8.2 Classpath Configuration

The GCS Event handler and the BigQuery Event handler use the Java SDK provided by Google. Google does not provide a direct link to download the SDK.

You can download the SDKs using the following maven co-ordinates:

Google Cloud Storage

```
<dependency>
  <groupId>com.google.cloud</groupId>
  <artifactId>google-cloud-storage</artifactId>
  <version>1.113.9</version>
</dependency>
```

To download the GCS dependencies, execute the following script `<OGGDIR>/DependencyDownloader/gcs.sh`.

BigQuery

```
<dependency>
  <groupId>com.google.cloud</groupId>
  <artifactId>google-cloud-bigquery</artifactId>
  <version>1.111.1</version>
</dependency>
```

To download the BigQuery dependencies, execute the following script `<OGGDIR>/DependencyDownloader/bigquery.sh`. For more information, see `gcs.sh` in [Dependency Downloader Scripts](#).

Set the path to the GCS and BigQuery SDK in the `gg.classpath` configuration parameter. For example: `gg.classpath=./gcs-deps/*:./bq-deps/*`.

For more information, see [Dependency Downloader Scripts](#).

10.2.22.2.8.3 Proxy Configuration

When the replicat process is run behind a proxy server, you can use the `jvm.bootoptions` property to set the proxy server configuration. For example: `jvm.bootoptions=-Dhttps.proxyHost=some-proxy-address.com -Dhttps.proxyPort=80`.

10.2.22.2.8.4 INSERTALLRECORDS Support

Stage and merge targets supports INSERTALLRECORDS parameter.

See [INSERTALLRECORDS](#) in *Reference for Oracle GoldenGate*. Set the INSERTALLRECORDS parameter in the Replicat parameter file (.prm). Set the INSERTALLRECORDS parameter in the Replicat parameter file (.prm)

Setting this property directs the Replicat process to use bulk insert operations to load operation data into the target table.

To process initial load trail files, set the INSERTALLRECORDS parameter in the Replicat parameter file (.prm). Setting this property directs the Replicat process to use bulk insert operations to load operation data into the target table. You can tune the batch size of bulk inserts using the `gg.handler.bq.maxFileSize` File Writer property. The default value is set to 1GB.

The frequency of bulk inserts can be tuned using the File Writer `gg.handler.bq.fileRollInterval` property, the default value is set to 3m (three minutes).

10.2.22.2.8.5 BigQuery Dataset and GCP ProjectId Mapping

The BigQuery Event handler maps the table schema name to the BigQuery dataset.

For three-part table names, the table catalog name is mapped to the GCP projectId.

If the tables use distinct catalog names, then the BigQuery datasets can reside in multiple GCP projects. The GCP service account key should have the required privileges to create datasets and tables in the respective

The table catalog name is mapped to the GCP projectId.

- [Three-Part Table Names](#)
- [Mapping Table](#)

10.2.22.2.8.5.1 Three-Part Table Names

If the tables use distinct catalog names, then the BigQuery datasets would reside in multiple GCP projects. The GCP service account key should have the required privileges in the respective GCP projects. See [BigQuery Permissions](#).

10.2.22.2.8.5.2 Mapping Table

Table 10-28 Mapping Table

MAP statement in the Replicat parameter file	BigQuery Dataset	GCP ProjectId
MAP SCHEMA1.*, TARGET "bq-project-1".*.*;	SCHEMA1	bq-project-1
MAP "bq-project-2".SCHEMA2.*, TARGET *.*.*;	SCHEMA2	bq-project-2
MAP SCHEMA3.*, TARGET *.*.*; SCHEMA3		The default projectId from the GCP service account key file or the configuration <code>gg.eventhandler.bq.projectId</code> .

10.2.22.2.8.6 BigQuery Iceberg Tables

This event handler supports automatic creation of Iceberg tables in BigQuery.

For more information, see <https://cloud.google.com/bigquery/docs/iceberg-tables#api>

Iceberg tables created by BigQuery uses Parquet file format, this cannot be modified.

A Cloud Storage URI and a Cloud Storage Connection are required to create Iceberg tables in BigQuery.

- [Cloud Storage URI](#)
- [Cloud Storage Connection](#)

10.2.22.2.8.6.1 Cloud Storage URI

The Cloud Storage URI is the location where the Iceberg table data is stored.

Every target Iceberg table requires a unique Cloud Storage URI.

For uniqueness, the value configured in the `gg.eventhandler.bq.storageUri` property is appended with the fully qualified table name to create the Iceberg table.

For example:

```
gg.eventhandler.bq.storageUri=gs://<bucket-name>/path/to/iceberg-tables/
<project-id>/<dataset-name>/<short-table-name>
```

10.2.22.2.8.6.2 Cloud Storage Connection

Follow the steps here: <https://cloud.google.com/bigquery/docs/create-cloud-resource-connection> to create a Cloud Storage Connection.

Ensure that the service account associated with the Cloud Storage Connection has the permissions to access the Cloud Storage URI.

The Cloud Storage Connection and the Cloud Storage bucket must reside in the same GCP region.

Note the Connection ID of the Cloud Storage Connection and ensure that the Connection ID must be prepended with the GCP region during configuration.

For example:

```
gg.eventhandler.bq.storageConnectionId=us-central1.<storage_connection_id>
```

10.2.22.2.8.7 End-to-End Configuration

The following is an end-end configuration example which uses auto configuration for File Writer (FW) handler, GCS, and BigQuery Event handlers.

This sample properties file is located at: `AdapterExamples/big-data/bigquery-via-gcs/bq.props`.

```
# Configuration to load GoldenGate trail operation records
# into Google Big Query by chaining
# File writer handler -> GCS Event handler -> BQ Event handler.
# Note: Recommended to only edit the configuration marked as TODO
# The property gg.eventhandler.gcs.credentialsFile need not be set if
# the GOOGLE_APPLICATION_CREDENTIALS environment variable is set.

gg.target=bq

## The GCS Event handler
#TODO: Edit the GCS bucket name
gg.eventhandler.gcs.bucketMappingTemplate=<gcs-bucket-name>
#TODO: Edit the GCS credentialsFile
gg.eventhandler.gcs.credentialsFile=/path/to/gcp/credentialsFile
```

```
## The BQ Event handler
## No mandatory configuration required.

#TODO: Edit to include the GCS Java SDK and BQ Java SDK.
gg.classpath=/path/to/gcs-deps/*:/path/to/bq-deps/*
#TODO: Edit to provide sufficient memory (at least 8GB).
jvm.bootoptions=-Xmx8g -Xms8g
#TODO: If running OGGBD behind a proxy server.
#jvm.bootoptions=-Xmx8g -Xms512m -Dhttps.proxyHost=<ip-address> -Dhttps.proxyPort=<port>
```

10.2.22.2.9 Troubleshooting and Diagnostics

- **DDL not applied on the target table:** Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) does not support DDL replication.
- **SQL Errors:** In case there are any errors while executing any SQL, the entire SQL statement along with the bind parameter values are logged into the GG for DAA handler log file.
- **Co-existence of the components:** The location/region of the machine where Replicat process is running and the BigQuery dataset/GCS bucket impacts the overall throughput of the apply process.
Data flow is as follows: **GoldenGate -> GCS bucket -> BigQuery**. For best throughput, ensure that the components are located as close as possible.
- `com.google.cloud.bigquery.BigQueryException: Access Denied: Project <any-gcp-project>: User does not have bigquery.datasets.create permission in project <any-gcp-project>. The service account key used by GG for DAA does not have permission to create datasets in this project. Grant the permission bigquery.datasets.create and restart the Replicat process. The privileges are listed in BigQuery Permissions.`

10.2.22.2.10 Restrictions on Database Object Names

- **Catalog (Project ID):** Must start with a lowercase letter, contain only lowercase letters, digits, or hyphens; cannot end with a hyphen; length 6-30; cannot contain restricted strings like "google", "ssl", "undefined", "null"; also avoids period (.), asterisk (*), question mark (?), double quote ("), backslash (\) and control characters (ASCII 0-31).
- **Schema (Dataset):** Must contain only ASCII letters, digits, underscores; length 1-1024; avoids common invalid characters like period (.), asterisk (*), question mark (?), double quote ("), backslash (\) and control characters (ASCII 0-31).
- **Table:** Must contain only allowed Unicode categories (letters, marks, numbers, connector/hyphen punctuation, spaces); length 1-1024; avoids common invalid characters like period (.), asterisk (*), question mark (?), double quote ("), backslash (\) and control characters (ASCII 0-31).
- **Specific limitation for GGDA 21c and previous releases:** Supports only names that comply with the Avro specification for identifier names: must start with `[A-Z a-z_]``, followed by `[A-Za-z0-9_]``
- Additionally, all object names are also limited by the supported list of characters for the respective target.

10.2.23 Google Cloud Storage

Topics:

- [Overview](#)
- [Prerequisites](#)
- [Buckets and Objects](#)
- [Authentication and Authorization](#)
- [Configuration](#)
- [Troubleshooting and Diagnostics](#)

10.2.23.1 Overview

Google Cloud Storage (GCS) is a service for storing objects in Google Cloud Platform. You can use the GCS Event handler to load files generated by the File Writer handler into GCS.

10.2.23.2 Prerequisites

Ensure to have the following set up:

- Google Cloud Platform (GCP) account set up.
- Google service account key with the relevant permissions.
- GCS Java Software Development Kit (SDK)

10.2.23.3 Buckets and Objects

Buckets are the basic containers in GCS that store data (objects). Objects are the individual pieces of data that you store in the Cloud Storage bucket.

10.2.23.4 Authentication and Authorization

A Google Cloud Platform (GCP) service account is a special kind of account used by an application, not by a person. Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) uses a service account key for accessing GCS service. You need to create a service account key with the relevant Identity and Access Management (IAM) permissions.

Use the JSON key type to generate the service account key file.

You can either set the path to the service account key file in the environment variable `GOOGLE_APPLICATION_CREDENTIALS` or in the GCS Event handler property `gg.eventhandler.name.credentialsFile`. You can also specify the individual keys of credentials file like `clientId`, `clientEmail`, `privateKeyId` and `privateKey` into corresponding handler properties instead of specifying the credentials file path directly. This enables the credential keys to be encrypted using Oracle wallet.

For more information about creating a service account key, see [GCP documentation](#). The following are the IAM permissions to be added into the service account used to run GCS Event handler.

- [Bucket Permissions](#)
- [Object Permissions](#)

10.2.23.4.1 Bucket Permissions

Table 10-29 Bucket Permissions

Bucket Permission Name	Description
storage.buckets .create	Create new buckets in a project.
storage.buckets .delete	Delete buckets.
storage.buckets .get	Read bucket metadata, excluding IAM policies.
storage.buckets .list	List buckets in a project. Also read bucket metadata, excluding IAM policies, when listing.
storage.buckets .update	Update bucket metadata, excluding IAM policies.

10.2.23.4.2 Object Permissions

Table 10-30 Object Permissions

Object Permission Name	Description
storage.objects .create	Add new objects to a bucket.
storage.objects .delete	Delete objects.
storage.objects .get	Read object data and metadata, excluding ACLs.
storage.objects .list	List objects in a bucket. Also read object metadata, excluding ACLs, when listing.
storage.objects .update	Update object metadata, excluding ACLs.

10.2.23.5 Configuration

Table 10-31 Object Permissions

Properties	Required /Optional	Legal Values	Default	Explanation
<code>gg.event.handler.name.type</code>	Required	<code>gcs</code>	None	Selects the GCS Event Handler for use with File Writer handler.
<code>gg.event.handler.name.location</code>	Optional	A valid GCS location.	None	If the GCS bucket does not exist, a new bucket will be created in this GCS location. If location is not specified, new bucket creation will fail. GCS location reference: GCS locations .

Table 10-31 (Cont.) Object Permissions

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.event.handler.name.bucketMappingTemplate</code>	Required	A string with resolvable keywords and constants used to dynamically generate a GCS bucket name.	None	A GCS bucket is created by the GCS Event handler if it does not exist using this name. See Bucket Naming Guidelines For more information about supported keywords, see Template Keywords .

Table 10-31 (Cont.) Object Permissions

Properties	Required/Optional	Legal Values	Default	Explanation
gg.event.handler.name.pathMappingTemplate	Required	A string with resolvable keywords and constants used to dynamically generate the path in the GCS bucket to write the file.	None	Use keywords interlaced with constants to dynamically generate a unique GCS path names at runtime. Example path name: ogg/data/\${groupName}/\${fullyQualifiedTableName}. For more information about supported keywords, see Template Keywords .

Table 10-31 (Cont.) Object Permissions

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.event.handler.name.fileNameMappingTemplate</code>	Optional	A string with resolvable keywords and constants used to dynamically generate a file name for the GCS object.	None	Use resolvable keywords and constants used to dynamically generate the GCS object file name. If not set, the upstream file name is used. For more information about supported keywords, see Template Keywords
<code>gg.event.handler.name.finalizeAction</code>	Optional	A unique string identifier cross referencing a child event handler.	No event handler configured.	Sets the downstream event handler that is invoked on the file roll event. A typical example would be use a downstream to load the GCS data into Google BigQuery using the BigQuery Event handler.

Table 10-31 (Cont.) Object Permissions

Properties	Required /Optional	Legal Values	Default	Explanation
gg.event.handler. name.credentials File	Optional	Relative or absolute path to the service account key file.	No	Sets the path to the service account key file. Alternatively, if the environment variable GOOGLE_APPLICATION_CREDENTIALS is set to the path to the service account key file, then you need not set this parameter.

Table 10-31 (Cont.) Object Permissions

Properties	Required/Optional	Legal Values	Default	Explanation
gg.event.handler.name.storageClasses	Optional	STANDARD NEARLINE COLDLINE ARCHIVE REGIONAL MULTI_REGIONAL DURABLE REDUCED_AVAILABILITY	None	The storage class you set for an object affects the object's availability and pricing model. If this property is not set, then the storage class for the file is set to the default storage class for the respective bucket. If the bucket does not exist and storage class is specified, then a new bucket is created with this storage class as its default.

Table 10-31 (Cont.) Object Permissions

Properties	Required/Optional	Legal Values	Default	Explanation
gg.event.handler.name.kmsKey	Optional	Key names in the format: projects / <PROJECT> / location s / <LOCATION> / keyRings / <RING_NAME> / cryptoKeys / <KEY_NAME>. <PROJECT>: Google project-id. <LOCATION>: Location of the GCS bucket. <RING_NAME>: Google Cloud KMS key ring name. <KEY_NAME>: Google Cloud KMS key name.	None	Google Cloud Storage always encrypts your data on the server side, before it is written to disk using Google-managed encryption keys. As an additional layer of security, customers may choose to use keys generated by Google Cloud Key Management Service (KMS). This property can be used to set a customer managed Cloud KMS key to encrypt GCS objects. When using customer managed keys, the gg.event.handler.name.concurrency property cannot be set to a

Table 10-31 (Cont.) Object Permissions

Propertie s	Required /Optional	Legal Values	Default	Explanati on
				value greater than one because with customer managed keys GCP does not allow multi-part uploads using object compositi on.

Table 10-31 (Cont.) Object Permissions

Properties	Required /Optional	Legal Values	Default	Explanation
gg.event.handler.name.concurrency	Optional	Any number in the range 1 to 32.	10	If concurrency is set to a value greater than one, then the GCS Event handler performs multi-part uploads using composition. The multi-part uploads spawn concurrent threads to upload each part. The individual parts are uploaded to the following directory <i><bucketMappingTemplate>/oggtmp</i> . This directory is reserved for use by Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA). This provides better throughput rates for uploading

Table 10-31 (Cont.) Object Permissions

Properties	Required/Optional	Legal Values	Default	Explanation
				large files. Multi-part uploads are used for files with size greater than 10 megabytes.
gg.event.handler. gcs.credentialsId	Optional	Valid Big Query Credentials Client Id	NA	Provides the client ID key from the credentials file for connecting to Google Big Query service account.
gg.event.handler. gcs.credentialsEmail	Optional	Valid Big Query Credentials Client Email	NA	Provides the client Email key from the credentials file for connecting to Google Big Query service account.
gg.event.handler. gcs.privateKeyId	Optional	Valid Big Query Credentials Client Email	NA	Provides the client Email key from the credentials file for connecting to Google Big Query service account.

Table 10-31 (Cont.) Object Permissions

Properties	Required/Optional	Legal Values	Default	Explanation
gg.event.handler.gcs.privateKey	Optional	Valid Big Query Credentials Private Key.	NA	Provides the Private Key from the credentials file for connecting to Google Big Query service account.
gg.event.handler.name.projectId	Optional	The Google project-id project-id associated with the service account.	NA	Sets the project-id of the Google Cloud project that houses the storage bucket. Auto configure will automatically configure this property by accessing the service account key file unless user wants to override this explicitly.

Table 10-31 (Cont.) Object Permissions

Properties	Required/Optional	Legal Values	Default	Explanation
gg.event.handler.name.url	Optional	A legal URL to connect to Google Cloud Storage including scheme, server name and port (if not the default port). The default is https://storage.googleapis.com.	https://storage.googleapis.com	Allows the user to set a URL for a private endpoint to connect to GCS.

Note

To be able to connect GCS to the Google Cloud Service account, ensure that either of the following is configured: the credentials file property with the relative or absolute path to credentials JSON file or the properties for individual credentials keys. The configuration property to individually add google service account credential key enables them to encrypt using the Oracle wallet.

- [Classpath Configuration](#)
- [Proxy Configuration](#)
- [Sample Configuration](#)

10.2.23.5.1 Classpath Configuration

The GCS Event handler uses the Java SDK for Google Cloud Storage. The classpath must include the path to the GCS SDK.

- [Dependencies](#)

10.2.23.5.1.1 Dependencies

You can download the SDK using the following maven co-ordinates:

```
<dependency>
  <groupId>com.google.cloud</groupId>
  <artifactId>google-cloud-storage</artifactId>
  <version>1.113.9</version>
</dependency>
```

Alternatively, you can download the GCS dependencies by running the script: `<OGGDIR>/DependencyDownloader/gcs.sh`.

Edit the `gg.classpath` configuration parameter to include the path to the GCS SDK.

10.2.23.5.2 Proxy Configuration

When the Replicat process runs behind a proxy server, you can use the `jvm.bootoptions` property to set proxy server configuration. For Example:

```
jvm.bootoptions=-Dhttps.proxyHost=some-proxy-address.com
-Dhttps.proxyPort=80
```

10.2.23.5.3 Sample Configuration

```
#The GCS Event handler
gg.eventhandler.gcs.type=gcs
gg.eventhandler.gcs.pathMappingTemplate=${fullyQualifiedTableName}
#TODO: Edit the GCS bucket name
gg.eventhandler.gcs.bucketMappingTemplate=<gcs-bucket-name>
#TODO: Edit the GCS credentialsFile
gg.eventhandler.gcs.credentialsFile=/path/to/gcs/credentials-file
gg.eventhandler.gcs.finalizeAction=none
gg.classpath=/path/to/gcs-deps/*
jvm.bootoptions=-Xmx8g -Xms8g
```

10.2.23.6 Troubleshooting and Diagnostics

Duplicate records after Replicat Recovery

Google Cloud Storage (GCS) handler replication uses File Writer Handler and GCS handler in the replicat. Oracle GoldenGate prioritizes no data loss and guarantees no data loss in case of failures by at least once semantics in GCS (json, csv, delimitedtext, avro_orc, parquet) delivery. In the cases if replicat runs fine and normally shut down, then exactly once is supported. In case of failures (because of network failures), there are various reason that can lead into duplicates in recovery.

Two cases where duplicates can occur:

1. If data is written and a failure occurs between when the data is written, and when the checkpoint is moved. Then upon restart the replicat backs up to the previous checkpoint and data can unfortunately be replayed.
2. The rolling of the data files occurs based on customer configured triggers. Trigger can be file size, time, inactivity, or time of day. The rolling does not necessarily happen on a transaction commit boundary. The trigger causes writing to the current file to complete, the post processing transformation and movement complete, and any state on that file is deleted. If a replicat abend occurs in between when the rolling is processed and when the checkpoint is moved, then upon restart, it can again replay those messages.

If you observe duplicate records in case of GCS replicat recovery, then it is an expected behavior. If you observe duplicates while replicat is running fine, then file a support ticket.

10.2.24 Google Cloud Managed Service for Apache Kafka

The Google Cloud Managed Service for Apache Kafka is a fully managed, Apache Kafka-compatible service on Google Cloud .

This chapter describes how to use the Google Cloud Managed Service for Apache Kafka.

- [Overview](#)

- [Configuration from Oracle Cloud Infrastructure \(OCI\)](#)
- [Sample Configuration](#)

10.2.24.1 Overview

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) can be used to publish change data capture operation messages to Google Cloud Managed Service for Apache Kafka.

GG for DAA connecting to Google Cloud Managed Service for Apache Kafka needs to run on GCP within the same subnet or, it can connect to Google Cloud Managed Service for Apache Kafka through ipSec VPN/ fastconnect.

Note

- Google Cloud Managed Service for Apache Kafka does not support public client access.
- GG for DAA Kafka Handler, see [Kafka Handler](#) or the Kafka Connect Handler, see [Kafka Connect Handler Schema registry](#), with Google Cloud Managed Service for Apache Kafka is not supported.

10.2.24.2 Configuration from Oracle Cloud Infrastructure (OCI)

To configure the Google Cloud Managed Service for Apache Kafka from Oracle Cloud Infrastructure (OCI):

1. Create Managed Kafka in Google Cloud with the subnet used in interconnect.
2. In GCS, go to Cloud DNS,
3. Click on the zone name created for Managed Kafka. It displays all the broken IP addresses that needs to be resolved by OCI.
4. In OCI, go to **Networking, DNS Management**, then **private views**.
5. Click on the **private views** created for the VCN used in OCI - GCP fastconnect.
6. Go to **private zones** and click **Create zone**.
7. For Zone name, provide managedkafka.<gcs-project-ids>.cloud.goog.
8. Click on the **Zone name**.
9. Go to **Records** and click **Manage records, Add record**. Add a record for each IP of your Managed Kafka brokers that you located in previous steps.
For Name, provide the complete DNS Name and for RData, provide the IP. For example:
DNS Name: bootstrap.mykafka.europe-west2.managedkafka.my-project.cloud.goog / Record Data: 172.20.0.3.
10. Use `base64 -w 0 < my_service_account_key.json > password.txt` to generate a password from service account key file.
11. Verify, if the target topic is accessible from the subnet in OCI.
In OCI VM, test with a kafka client: `./kafka-console-producer.sh --bootstrap-server bootstrap.mykafka.europe-west2.managedkafka.my-project.cloud.goog:9092 --topic testtopic --producer.config /home/opc/kfk.properties` Message write should be success.
12. Configure Oracle GoldenGate Kafka replicat and run the replicat.

10.2.24.3 Sample Configuration

The following is a sample configuration for the Google Cloud Managed Service for Apache Kafka :

```
bootstrap.servers=bootstrap.mykafka.europe-west2.managedkafka.my-project.cloud.goog:9092
value.serializer=org.apache.kafka.common.serialization.ByteArraySerializer
key.serializer=org.apache.kafka.common.serialization.ByteArraySerializer
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="myuser" password="mypassword"
```

① Note

- Username is your GCP SERVICE_ACCOUNT_EMAIL_ADDRESS.
- Password is CONTENTS_OF_BASE64_ENCODED_PASSWORD_FILE from the previous steps where you have used the `base64 -w 0 < my_service_account_key.json > password.txt` to generate a password from service account key file..

Related Topics

- [Google Cloud Managed Service for Apache Kafka - Configure SASL authentication](#)

10.2.25 Google Pub/Sub

- [Overview](#)
- [Detailed Functionality](#)
- [Setting up and Running the Google PubSub Handler](#)
- [Configuring Handler Authentication](#)
- [Google PubSub Handler Configuration](#)
- [Proxy Settings](#)
- [Sample Configuration](#)
- [Google PubSub Dependencies](#)

10.2.25.1 Overview

Pub/Sub allows services to communicate asynchronously, with latencies on the order of 100 milliseconds.

Pub/Sub enables you to create systems of event producers and consumers, called **publishers** and **subscribers**. Publishers communicate with subscribers asynchronously by broadcasting events, rather than by synchronous remote procedure calls (RPCs).

Publishers send events to the Pub/Sub service, without regard to how or when these events are to be processed. Pub/Sub then delivers events to all the services that react to them. In systems communicating through RPCs, publishers must wait for subscribers to receive the

data. However, the asynchronous integration in Pub/Sub increases the flexibility and robustness of the overall system.

10.2.25.2 Detailed Functionality

Transaction Mode

The following configuration sets the Google PubSub Handler to transaction mode:

`gg.handler.name.Mode=tx`. In transaction mode, the serialized data is concatenated for every operation in a transaction from the source Oracle GoldenGate trail files. The contents of the concatenated operation data is the value of the Google Pubsub message. The result is that the messages comprise data from 1 to N operations, where N is the number of operations in the transaction.

Operation Mode

The following configuration sets the GooglePubSub Handler to operation mode:

`gg.handler.name.Mode=op`.

In operation mode, the serialized data for each operation is placed into an individual Google Pubsub message as the value. This means that there is a 1 to 1 relationship between the incoming operations and the number of Google Pubsub messages produced.

10.2.25.3 Setting up and Running the Google PubSub Handler

Configuring Classpath

For the Google PubSub Handler to connect to Google PubSub and run, the properties file and the Google PubSub client JARs must be configured in the `gg.classpath` configuration variable. For the GooglePubSub jar files use the dependency downloader tool.

The recommended storage location for the Google PubSub properties file is the Oracle GoldenGate `dirprm` directory. The `gg.classpath` must be configured precisely.

The path of the Google PubSub properties file must contain the path with no wildcard appended. If the `*` wildcard is included in the path to the properties file, then the file is not picked up. Conversely, path to the dependency JARs must include the `*` wild card character in order to include all the JAR files in that directory in the associated classpath. Do not use `*.jar`. The following is an example of the correctly configured classpath: `gg.classpath={google PubSub install dir}/libs/`.

10.2.25.4 Configuring Handler Authentication

You have to configure the Google PubSub Handler authentication using the credentials in the JSON file downloaded from Google PubSub.

Download the credentials file:

1. Login into your Google account at cloud.google.com.
2. Click **Console**, and then to go to the Dashboard where you can select your project.
3. From the navigation menu, click **APIs & Services**, and then select **Credentials**.
4. From the **Create Credentials** menu, choose Service account key.
5. Choose the **JSON key** type to download the JSON credentials file for your system.

Specify the path to the credentials file in the properties file with the `gg.handler.name.credentialsFile` configuration property.

The path of the credentials file must contain the path with no wildcard appended. If you include the `*` wildcard in the path to the credentials file, then the file is not recognized.

Google PubSub credentials file configuration sample:

```
{
  "type": "<accountType>",
  "project_id": "<projectType>",
  "private_key_id": "<privateKeyId>",
  "private_key": "<privateKey>",
  "client_email": "<googleCloudSubscriptionEmailId>",
  "client_id": "<subscriptionClientId>",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "",
  "client_x509_cert_url": ""
}
```

10.2.25.5 Google PubSub Handler Configuration

Property Name	Required/Optional	Property Value	Default	Description
gg.handlerlist	Required	name (choice of any name)	None	List of handlers to be used
gg.handler.name.type	Required	oracle.goldengate.handler.googlepubsub.GooglePubsubHandler	None	Type of handler to use
gg.handler.name.format	Optional	Formatter class or short code	json	Formatter to use to format payload. Can be one of xml, delimitedtext, json, json_row, avro_row, or avro_op
gg.handler.name.credentialsFile	Required	The name of the credentials file in json format with absolute path	json	NA
gg.handler.name.mode	Optional	tx/op	tx	NA
gg.handler.name.topicMappingTemplate	Required	NA	NA	Google PubSub Topic name to which the replicat will publish messages to. See Using Templates to Resolve the Topic Name and Message Key
gg.handler.name.projectName	Required	NA	NA	Google PubSub project name
gg.handler.name.batchProcessing	Optional	true/false	true	When enabled the messages will be processed in batches enhancing the performance replicat.

<code>gg.handler.name.requestBytesThreshold</code>	Optional	Numeric integer value	10485760 (bytes)	Maximum size of publish request by specifying the maximum number of bytes
<code>gg.handler.name.messageCountBatchSize</code>	Optional	Numeric integer value	1000	Maximum size of publish request by specifying the maximum number of messages
<code>gg.handler.name.publishDelayThreshold</code>	Optional	Numeric integer value	1 (ms)	Provides flexibility to control how long to wait before sending a batch, specifically in regard to the amount of time messages are held in order to fill batches. Decreasing this value improves latency. This property value is the batch-time interval in milliseconds.
<code>gg.handler.name.attributesTemplate</code>	Optional	Comma delimited list of attribute keywords.	None	The different properties of the message can be set as attributes to the google pubsub message. This facilitates the user/ subscriber to filter messages based on the set attributes. See Metacolumn Keywords for more information about configuring this property.
<code>gg.handler.name.orderingKeyMappingTemplate</code>	Optional	A template string value to resolve the pubsub message ordering key at runtime. When running transaction mode use a static orderingKey to order the messages.	None	See Using Templates to Resolve the Topic Name and Message Key

<code>gg.handler.name.flowControl</code>	Optional	true/false	false	When enabled it sets the limit of publish requests to accumulate before blocking further publish requests from queuing up.
<code>gg.handler.name.fcMaxOutstandingElementCount</code>	Optional	Numeric integer value	10000	Max number of publish requests to queue before starting to block any further requests.
<code>gg.handler.name.fcMaxOutstandingRequestBytes</code>	Optional	Numeric integer value	524288000 (500MB)	Maximum number of request bytes to queue before starting to block any further requests.

10.2.25.6 Proxy Settings

To connect to Google PubSub using a proxy server, you must configure the proxy host and the proxy port in the properties file as follows:

```
jvm.bootoptions= -Dhttps.proxyHost=proxy_host_name -
Dhttps.proxyPort=proxy_port_number
```

10.2.25.7 Sample Configuration

The following is a sample configuration for the Google PubSub Handler:

```
gg.handlerlist=pubsub
#The Google Pub/Sub Handler
gg.handler.pubsub.type=googlepubsub
gg.handler.pubsub.mode=op
#Set the path to the JSON credentials file
gg.handler.pubsub.credentialsFile=
#Set the template to resolve the topic name
gg.handler.pubsub.topicMappingTemplate=
#Set the project name
gg.handler.pubsub.projectName=
#Set the template to resolve the order key
gg.handler.pubsub.orderingKeyMappingTemplate=
gg.handler.pubsub.format=json_row
gg.handler.pubsub.format.metaColumnsTemplate=${objectname[table]},${optype[op_type]},$
{timestamp[op_ts]},${currenttimestamp[current_ts]},${position[pos]}
```

10.2.25.8 Google PubSub Dependencies

The Google PubSub client libraries are required for integration with Google PubSub. The maven coordinates are as follows:

Maven groupId: com.google.cloud

Maven artifactId: google-cloud-pubsub

Version: 1.129.3

```
api-common-2.31.0.jar
guava-33.1.0-jre.jar
protobuf-java-util-3.25.3.jar
listenablefuture-9999.0-empty-to-avoid-conflict-with-guava.jar
proto-google-iam-v1-1.34.0.jar
threetenbp-1.6.9.jar
google-cloud-pubsub-1.129.3.jar
opencensus-proto-0.2.0.jar
google-http-client-gson-1.44.1.jar
grpc-protobuf-lite-1.62.2.jar
grpc-util-1.62.2.jar
javax.annotation-api-1.3.2.jar
checker-qual-3.42.0.jar
error_prone_annotations-2.26.1.jar
gax-grpc-2.48.0.jar
httpcore-4.4.16.jar
proto-google-common-protos-2.39.0.jar
google-auth-library-oauth2-http-1.23.0.jar
opencensus-contrib-http-util-0.31.1.jar
protobuf-java-3.25.3.jar
proto-google-cloud-pubsub-v1-1.111.3.jar
grpc-api-1.62.2.jar
perfmak-api-0.27.0.jar
gax-2.48.0.jar
jsr305-3.0.2.jar
conscrypt-openjdk-uber-2.5.2.jar
failureaccess-1.0.2.jar
grpc-inprocess-1.62.2.jar
grpc-grpclb-1.62.2.jar
grpc-netty-shaded-1.62.2.jar
google-http-client-1.44.1.jar
grpc-core-1.62.2.jar
j2objc-annotations-3.0.0.jar
commons-codec-1.16.1.jar
gson-2.10.1.jar
grpc-auth-1.62.2.jar
google-auth-library-credentials-1.23.0.jar
grpc-services-1.62.2.jar
grpc-context-1.62.2.jar
animal-sniffer-annotations-1.23.jar
opencensus-api-0.31.1.jar
gax-httpjson-2.48.0.jar
auto-value-annotations-1.10.4.jar
grpc-stub-1.62.2.jar
annotations-4.1.1.4.jar
grpc-xds-1.62.2.jar
grpc-alts-1.62.2.jar
grpc-googleapis-1.62.2.jar
httpclient-4.5.14.jar
re2j-1.7.jar
grpc-protobuf-1.62.2.jar
```

10.2.26 Java Message Service (JMS)

The Java Message Service (JMS) Handler allows operations from a trail file to be formatted in messages, and then published to JMS providers like Oracle Weblogic Server, Websphere, and ActiveMQ.

This chapter describes how to use the JMS Handler.

- [Overview](#)
- [Setting Up and Running the JMS Handler](#)
- [JMS Dependencies](#)

10.2.26.1 Overview

The Java Message Service is a Java API that allows applications to create, send, receive, and read messages. The JMS API defines a common set of interfaces and associated semantics that allow programs written in the Java programming language to communicate with other messaging implementations.

The JMS Handler captures the Oracle GoldenGate trail and sends those messages to the configured JMS providers.

Note

The Java Message Service (JMS) Handler does not support DDL operations. In case of DDL operations, replicat/extract is expected to fail.

10.2.26.2 Setting Up and Running the JMS Handler

The JMS Handler setup (JNDI configuration) depends on the JMS provider that you use.

The following sections provide instructions for configuring the JMS Handler components and running the handler.

Runtime Prerequisites

The JMS provider should be up and running with the required ConnectionFactory and QueueConnectionFactory and TopicConnectionFactory configured.

Security

Configure the SSL according to the JMS Provider used.

- [Classpath Configuration](#)
Oracle recommends that you store the JMS Handler properties file in the Oracle GoldenGate dirprm directory.
- [Java Naming and Directory Interface Configuration](#)
- [Topic/Queue Name Selection](#)
- [Handler Configuration](#)
- [Sample Configuration Using Oracle WebLogic Server](#)

10.2.26.2.1 Classpath Configuration

Oracle recommends that you store the JMS Handler properties file in the Oracle GoldenGate dirprm directory.

The JMS Handler requires the JMS Provider client JARs are in the classpath in order to execute. Additionally, in Java 8, the Java EE Specification classes have been moved out of the JDK to an independent project. JMS is a part of the Java EE Specification so the Java EE Specification jar is an additional dependency. For more information to download the jar, see [JMS Dependencies](#).

The location of the providers client JARs is similar to:

```
gg.classpath= path_to_the_providers_client_jars
```

10.2.26.2.2 Java Naming and Directory Interface Configuration

You configure the Java Naming and Directory Interface (JNDI) properties to connect to an Initial Context to look up the connection factory and initial destination.

Table 10-32 JNDI Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
java.naming.provider.url	Required	Valid provider URL with port	None	Specifies the URL that the handler uses to look up objects on the server. For example, <code>t3://localhost:7001</code> or if SSL is enabled <code>t3s://localhost:7002</code> .
java.naming.factory.initial	Required	Initial Context factory class name	None	Specifies which initial context factory to use when creating a new initial context object. For Oracle WebLogic Server, the value is <code>weblogic.jndi.WLInitialContextFactory</code> .
java.naming.security.principal	Required	Valid user name	None	Specifies the user name to use.
java.naming.security.credentials	Required	Valid password	None	Specifies the password for the user.

10.2.26.2.3 Topic/Queue Name Selection

The destination is resolved at runtime using this configuration parameter: `gg.handler.name.destinationTemplate`. You can configure a static string, keywords, or a combination of static strings and keywords to dynamically resolve the topic/queue name at runtime based on the context of the current operation, see [Using Templates to Resolve the Topic Name and Message Key](#) for sample templates.

The replicat will autocreate the target Queue/Topic if not already present.

Note

Auto creation of Queue/Topic is not supported for the Oracle WebLogic Server.

10.2.26.2.4 Handler Configuration

You configure the JMS Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the JMS Handler, you must first configure the handler type by specifying `gg.handler.name.type=jms` and the other JMS properties as follows:

Table 10-33 JMS Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.type</code>	Required	JMS	None	Set to <code>jms</code> to send transactions, operations, and metadata as formatted text messages to a JMS provider. Set to <code>jms_map</code> to send JMS map messages.
<code>gg.handler.name.destinationTemplate</code>	Required	Valid queue or topic name template	None	Sets the queue or topic to which the message is sent. This must be correctly configured on the JMS server. For example, for a static queue/topic <code>queue/A</code> , <code>queue.Test</code> , <code>example.MyTopic</code> . For example, <code>queue/A</code> , <code>queue.Test</code> , <code>example.MyTopic</code> .
<code>gg.handler.name.destinationType</code>	Optional	<code>queue</code> <code>topic</code>	<code>queue</code>	Specifies whether the handler is sending to a queue (a single receiver) or a topic (publish/subscribe). The <code>gg.handler.name.queueOrTopic</code> property is an alias of this property. Set to <code>queue</code> removes a message from the queue once it has been read. Set to <code>topic</code> publishes messages and can be delivered to multiple subscribers.
<code>gg.handler.name.connectionFactory</code>	Required	Valid connection factory name	None	Specifies the name of the connection factory to lookup using JNDI. The <code>gg.handler.name.ConnectionFactoryJNDIName</code> property is an alias of this property. .
<code>gg.handler.name.useJndi</code>	Optional	<code>true</code> <code>false</code>	<code>true</code>	Set to <code>false</code> , then JNDI is not used to configure the JMS client. Instead, factories and connections are explicitly constructed.
<code>gg.handler.name.connectionUrl</code>	Optional	Valid connection URL	None	Specify only when you are not using JNDI to explicitly create the connection.
<code>gg.handler.name.connectionFactoryClass</code>	Optional	Valid connectionFactoryClass	None	Set to access a factory only when not using JNDI. The value of this property is the Java class name to instantiate, which constructs a factory object explicitly.
<code>gg.handler.name.physicalDestination</code>	Optional	Name of the queue or topic object obtained through the ConnectionFactory API instead of the JNDI provider	None	The physical destination is important when JMS is configured to use JNDI. The ConnectionFactory is resolved through a JNDI lookup. Setting the physical destination means that the queue or topic is resolved by invoking a method on the ConnectionFactory instead of invoking JNDI.
<code>gg.handler.name.user</code>	Optional	Valid user name	None	The user name to send messages to the JMS server.

Table 10-33 (Cont.) JMS Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.password</code>	Optional	Valid password	None	The password to send messages to the JMS server.
<code>gg.handler.name.sessionMode</code>	Optional	auto client dupso	auto	Sets the JMS session mode, these values equate to the standard JMS values: Session.AUTO_ACKNOWLEDGE The session automatically acknowledges a client's receipt of a message either when the session has successfully returned from a call to receive or when the message listener the session has called to process the message successfully returns. Session.CLIENT_ACKNOWLEDGE The client acknowledges a consumed message by calling the message's acknowledge method. Session.DUPS_OK_ACKNOWLEDGE This acknowledgment mode instructs the session to lazily acknowledge the delivery of messages.
<code>gg.handler.name.localTX</code>	Optional	true false	true	Sets whether local transactions are used when sending messages. Local transactions are enabled by default, unless sending and committing single messages one at a time. Set to false to disable local transactions.
<code>gg.handler.name.persistent</code>	Optional	true false	true	Sets the delivery mode to persistent or not. If you want the messages to be persistent, the JMS provider must be configured to log the message to stable storage as part of the client's send operation.
<code>gg.handler.name.priority</code>	Optional	Valid integer between 0-10	4	The JMS server defines a 10 level priority value, with 0 as the lowest and 9 as the highest.
<code>gg.handler.name.timeToLive</code>	Optional	Time in milliseconds	0	Sets the length of time in milliseconds from its dispatch time that a produced message is retained by the message system. Set to zero specifies that the time is unlimited.

Table 10-33 (Cont.) JMS Handler Configuration Properties

Properties	Require d/ Optiona l	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.custom</code>	Optional	Class names implem ent ing <code>oracle.g oldengat e.messag ing.hand ler.GGMe ssageLif eCycleLi stener</code>	None	Configures a message listener allowing properties to be set on the message before it is delivered.

Table 10-33 (Cont.) JMS Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.format</code>	Optional	<code>xml</code> <code>tx2ml</code> <code>xml2</code> <code>minxml</code> <code>csv</code> <code>fixed</code> <code>text</code> <code>logdump</code> <code>json</code> <code>json_op</code> <code>json_row</code> <code>delimite</code> <code>dtext</code> Velocity template	<code>delimite</code> <code>dtext</code>	Specifies the format used to transform operations and transactions into messages sent to the JMS server. The velocity template should point to the location of the template file. Samples are available under: <code>AdapterExamples/java-delivery/sample-dirprm/</code> . Example: <code>format_op2xml.vm</code> <pre> <\$op.TableName sqlType='\$op.sqlType' opType='\$op.opType' txInd='\$op.txState' ts='\$op.Timestamp' numCols='\$op.NumColumns' pos='\$op.Position'> #foreach(\$col in \$op) #if(! \$col.isMissing()) <\$col.Name colIndex='\$col.Index'> #if(\$col.hasBefore()) #if(\$col.isBeforeNull()) <before><isNull/></before> #else <before><![CDATA[\$col.before]]></before> #{end}## if col has 'before' value #{end}## if col 'before' is null #if(\$col.hasValue()) #if(\$col.isNull()) <after><isNull/></after> #{else} <after><![CDATA[\$col.value]]></after> #{end}## if col is null #{end}## if col has value </\$col.Name> #{end}## if column is not missing #{end}## for loop over columns </\$op.TableName> </pre>

Table 10-33 (Cont.) JMS Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.includeTables</code>	Optional	List of valid table names	None	Specifies a list of tables the handler will include. If the schema (or owner) of the table is specified, then only that schema matches the table name. Otherwise, the table name matches any schema. A comma separated list of tables can be specified. For example, to have the handler only process tables <code>foo.customer</code> and <code>bar.orders</code>. If the catalog and schema (or owner) of the table are specified, then only that catalog and schema matches the table name. Otherwise, the table name matches any catalog and schema. A comma separated list of tables can be specified. For example, to have the handler only process tables <code>dbo.foo.customer</code> and <code>dbo.bar.orders</code>. If any table matches the include list of tables, the transaction is included. The list of table names specified are case sensitive.
<code>gg.handler.name.excludeTables</code>	Optional	List of valid table names	None	Specifies a list of tables the handler will exclude. To selectively process operations on a table by table basis, the handler must be processing in operation mode. If the handler is processing in transaction mode, then when a single transaction contains several operations spanning several tables. If any table matches the exclude list of tables, the transaction is excluded. The list of table names specified are case sensitive.
<code>gg.handler.name.mode</code>	Optional	<code>op tx</code>	<code>op</code>	Specifies whether to output one operation per message (<code>op</code>) or one transaction per message (<code>tx</code>).
<code>gg.handler.name.metaHeadersTemplate</code>	Optional	Comma delimited list of metacolumn keywords.	None	Allows you to select metacolumns to inject context-based key value pairs into JMS message header properties using the metacolumn keyword syntax. JMS metacolumn Headers are not supported in transactional mode. Example: <pre>gg.handler.sample_jms.metaHeadersTemplate=\${primarykeys[JMSXGroupID]}</pre>

10.2.26.2.5 Sample Configuration Using Oracle WebLogic Server

```
#JMS Handler Template
gg.handlerlist=jms
gg.handler.jms.type=jms
#TODO: Set the message formatter type
gg.handler.jms.format=
#TODO: Set the destination for resolving the queue/topic name.
gg.handler.jms.destinationTemplate=

#Start of JMS handler properties when JNDI is used.
gg.handler.jms.useJndi=true
#TODO: Set the connectionFactory for resolving the queue/topic name.
gg.handler.jms.connectionFactory=
#TODO: Set the standard JNDI properties url, initial factory name,
principal and credentials.
java.naming.provider.url=
java.naming.factory.initial=
java.naming.security.principal=
java.naming.security.credentials=
End of JMS handler properties when JNDI is used.

#Start of JMS handler properties when JNDI is not used.
#TODO: Comment the above properties related to useJndi is true.
#TODO: Uncomment the below properties to configure when useJndi is false.
#gg.handler.jms.useJndi=false
#TODO: Set connectionURL of MQ.
#gg.handler.jms.connectionUrl=
#TODO: Set the connection Factory Class of the MQ.
#gg.handler.jms.connectionFactoryClass=

#TODO: Set the path the jms client library wlthint3client.jar
gg.classpath=
jvm.bootoptions=-Xmx512m -Xms32m
```

10.2.26.3 JMS Dependencies

The Java EE Specification APIs have moved out of the JDK in Java 8. JMS is a part of this specification, and therefore this dependency is required.

Maven groupId: javax

Maven artifactId: javaee-api

Version: 8.0

You can download the jar from [Maven Central Repository](#).

- [JMS 8.0](#)

10.2.26.3.1 JMS 8.0

javaee-api-8.0.jar

10.2.27 Java Database Connectivity

Learn how to use the Java Database Connectivity (JDBC) Handler, which can replicate source transactional data to a target or database.

This chapter describes how to use the JDBC Handler.

- [Overview](#)
- [Detailed Functionality](#)
The JDBC Handler replicates source transactional data to a target or database by using a JDBC interface.
- [Setting Up and Running the JDBC Handler](#)
Use the JDBC Metadata Provider with the JDBC Handler to obtain column mapping features, column function features, and better data type mapping.
- [Sample Configurations](#)

10.2.27.1 Overview

The Generic Java Database Connectivity (JDBC) Handler lets you replicate source transactional data to a target system or database by using a JDBC interface. You can use it with targets that support JDBC connectivity.

You can use the JDBC API to access virtually any data source, from relational databases to spreadsheets and flat files. JDBC technology also provides a common base on which the JDBC Handler was built. The JDBC handler with the JDBC metadata provider also lets you use Replicat features such as column mapping and column functions. For more information about using these features, see [Metadata Providers](#)

For more information about using the JDBC API, see <http://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/index.html>.

10.2.27.2 Detailed Functionality

The JDBC Handler replicates source transactional data to a target or database by using a JDBC interface.

- [Single Operation Mode](#)
- [Oracle Database Data Types](#)
- [MySQL Database Data Types](#)
- [Netezza Database Data Types](#)
- [Redshift Database Data Types](#)

10.2.27.2.1 Single Operation Mode

The JDBC Handler performs SQL operations on every single trail record (row operation) when the trail record is processed by the handler. The JDBC Handler does not use the BATCHSQL feature of the JDBC API to batch operations.

10.2.27.2.2 Oracle Database Data Types

The following column data types are supported for Oracle Database targets:

NUMBER
DECIMAL
INTEGER
FLOAT
REAL
DATE
TIMESTAMP
INTERVAL YEAR TO MONTH
INTERVAL DAY TO SECOND
CHAR
VARCHAR2
NCHAR
NVARCHAR2
RAW
CLOB
NCLOB
BLOB
TIMESTAMP WITH TIMEZONE¹
TIME WITH TIMEZONE²

10.2.27.2.3 MySQL Database Data Types

The following column data types are supported for MySQL Database targets:

INT
REAL
FLOAT
DOUBLE
NUMERIC
DATE
DATETIME
TIMESTAMP
TINYINT
BOOLEAN
SMALLINT
BIGINT
MEDIUMINT
DECIMAL
BIT
YEAR
ENUM
CHAR
VARCHAR

10.2.27.2.4 Netezza Database Data Types

The following column data types are supported for Netezza database targets:

¹ Time zone with a two-digit hour and a two-digit minimum offset.

² Time zone with a two-digit hour and a two-digit minimum offset.

byteint
smallint
integer
bigint
numeric(p,s)
numeric(p)
float(p)
Real
double
char
varchar
nchar
nvarchar
date
time
Timestamp

10.2.27.2.5 Redshift Database Data Types

The following column data types are supported for Redshift database targets:

SMALLINT
INTEGER
BIGINT
DECIMAL
REAL
DOUBLE
CHAR
VARCHAR
DATE
TIMESTAMP

10.2.27.3 Setting Up and Running the JDBC Handler

Use the JDBC Metadata Provider with the JDBC Handler to obtain column mapping features, column function features, and better data type mapping.

The following topics provide instructions for configuring the JDBC Handler components and running the handler.

- [Java Classpath](#)
- [Handler Configuration](#)
- [Statement Caching](#)
- [Setting Up Error Handling](#)

10.2.27.3.1 Java Classpath

The JDBC Java Driver location must be included in the class path of the handler using the `gg.classpath` property.

For example, the configuration for a MySQL database could be:

```
gg.classpath= /path/to/jdbc/driver/jar/mysql-connector-java-5.1.39-bin.jar
```

10.2.27.3.2 Handler Configuration

You configure the JDBC Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the JDBC Handler, you must first configure the handler type by specifying `gg.handler.name.type=jdbc` and the other JDBC properties as follows:

Table 10-34 JDBC Handler Configuration Properties

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.handler.name.type</code>	Required	jdbc	None	Selects the JDBC Handler for streaming change data capture into name.
<code>gg.handler.name.connectionURL</code>	Required	A valid JDBC connection URL	None	The target specific JDBC connection URL.
<code>gg.handler.name.DriverClass</code>	Target database dependent.	The target specific JDBC driver class name	None	The target specific JDBC driver class name.
<code>gg.handler.name.userName</code>	Target database dependent.	A valid user name	None	The user name used for the JDBC connection to the target database.
<code>gg.handler.name.password</code>	Target database dependent.	A valid password	None	The password used for the JDBC connection to the target database.
<code>gg.handler.name.maxActiveStatements</code>	Optional	Unsigned integer	Target database dependent	If this property is not specified, the JDBC Handler queries the target dependent database metadata indicating maximum number of active prepared SQL statements. Some targets do not provide this metadata so then the default value of 256 active SQL statements is used. If this property is specified, the JDBC Handler will not query the target database for such metadata and use the property value provided in the configuration. In either case, when the JDBC handler finds that the total number of active SQL statements is about to be exceeded, the oldest SQL statement is removed from the cache to add one new SQL statement.
<code>gg.mdp.connectionRetries</code>	Optional	Integer value	3	Specifies the number of times connections to the target data warehouse will be retried.

Table 10-34 (Cont.) JDBC Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.snowflake.connectionRetryIntervalSeconds</code>	Optional	Integer value	30	Specifies the delay in minutes between connection retry attempts.

10.2.27.3.3 Statement Caching

To speed up DML operations, JDBC driver implementations typically allow multiple statements to be cached. This configuration avoids repreparing a statement for operations that share the same profile or template.

The JDBC Handler uses statement caching to speed up the process and caches as many statements as the underlying JDBC driver supports. The cache is implemented by using an LRU cache where the key is the profile of the operation (stored internally in the memory as an instance of `StatementCacheKey` class), and the value is the `PreparedStatement` object itself.

A `StatementCacheKey` object contains the following information for the various DML profiles that are supported in the JDBC Handler:

DML operation type	<code>StatementCacheKey</code> contains a tuple of:
INSERT	(table name, operation type, ordered after-image column indices)
UPDATE	(table name, operation type, ordered after-image column indices)
DELETE	(table name, operation type)
TRUNCATE	(table name, operation type)

10.2.27.3.4 Setting Up Error Handling

The JDBC Handler supports using the `REPERROR` and `HANDLECOLLISIONS` Oracle GoldenGate parameters.

You must configure the following properties in the handler properties file to define the mapping of different error codes for the target database.

`gg.error.duplicateErrorCodes`

A comma-separated list of error codes defined in the target database that indicate a duplicate key violation error. Most of the drivers of the JDBC drivers return a valid error code so, `REPERROR` actions can be configured based on the error code. For example:

```
gg.error.duplicateErrorCodes=1062,1088,1092,1291,1330,1331,1332,1333
```

`gg.error.notFoundErrorCodes`

A comma-separated list of error codes that indicate missed DELETE or UPDATE operations on the target database.

In some cases, the JDBC driver errors occur when an UPDATE or DELETE operation does not modify any rows in the target database so, no additional handling is required by the JDBC Handler.

Most JDBC drivers do not return an error when a DELETE or UPDATE is affecting zero rows so, the JDBC Handler automatically detects a missed UPDATE or DELETE operation and triggers an error to indicate a not-found error to the Replicat process. The Replicat process can then execute the specified REPERROR action.

The default error code used by the handler is zero. When you configure this property to a non-zero value, the configured error code value is used when the handler triggers a not-found error. For example:

```
gg.error.notFoundErrorCodes=1222
```

gg.error.deadlockErrorCodes

A comma-separated list of error codes that indicate a deadlock error in the target database. For example:

```
gg.error.deadlockErrorCodes=1213
```

Setting Codes

Oracle recommends that you set a non-zero error code for the `gg.error.duplicateErrorCodes`, `gg.error.notFoundErrorCodes`, and `gg.error.deadlockErrorCodes` properties because Replicat does not respond to REPERROR and HANDLECOLLISIONS configuration when the error code is set to zero.

Sample Oracle Database Target Error Codes

```
gg.error.duplicateErrorCodes=1  
gg.error.notFoundErrorCodes=0  
gg.error.deadlockErrorCodes=60
```

Sample MySQL Database Target Error Codes

```
gg.error.duplicateErrorCodes=1022,1062  
gg.error.notFoundErrorCodes=1329  
gg.error.deadlockErrorCodes=1213,1614
```

10.2.27.4 Sample Configurations

The following topics contain sample configurations for the databases supported by the JDBC Handler from the Java Adapter properties file.

- [Sample Oracle Database Target](#)
- [Sample Oracle Database Target with JDBC Metadata Provider](#)
- [Sample MySQL Database Target](#)
- [Sample MySQL Database Target with JDBC Metadata Provider](#)

10.2.27.4.1 Sample Oracle Database Target

```
gg.handlerlist=jdbcwriter  
gg.handler.jdbcwriter.type=jdbc
```

```
#Handler properties for Oracle database target
gg.handler.jdbcwriter.DriverClass=oracle.jdbc.driver.OracleDriver
gg.handler.jdbcwriter.connectionURL=jdbc:oracle:thin:@<DBServer
address>:1521:<database name>
gg.handler.jdbcwriter.userName=<dbuser>
gg.handler.jdbcwriter.password=<dbpassword>
gg.classpath=/path/to/oracle/jdbc/driver/ojdbc5.jar
goldengate.userexit.timestamp=utc
goldengate.userexit.writers=javawriter
javawriter.stats.display=TRUE
javawriter.stats.full=TRUE
gg.log=log4j
gg.log.level=INFO
gg.report.time=30sec
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/
ggjava.jar:./dirprm
```

10.2.27.4.2 Sample Oracle Database Target with JDBC Metadata Provider

```
gg.handlerlist=jdbcwriter
gg.handler.jdbcwriter.type=jdbc

#Handler properties for Oracle database target with JDBC Metadata provider
gg.handler.jdbcwriter.DriverClass=oracle.jdbc.driver.OracleDriver
gg.handler.jdbcwriter.connectionURL=jdbc:oracle:thin:@<DBServer
address>:1521:<database name>
gg.handler.jdbcwriter.userName=<dbuser>
gg.handler.jdbcwriter.password=<dbpassword>
gg.classpath=/path/to/oracle/jdbc/driver/ojdbc5.jar
#JDBC Metadata provider for Oracle target
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:oracle:thin:@<DBServer address>:1521:<database name>
gg.mdp.DriverClassName=oracle.jdbc.driver.OracleDriver
gg.mdp.UserName=<dbuser>
gg.mdp.Password=<dbpassword>
goldengate.userexit.timestamp=utc
goldengate.userexit.writers=javawriter
javawriter.stats.display=TRUE
javawriter.stats.full=TRUE
gg.log=log4j
gg.log.level=INFO
gg.report.time=30sec
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/
ggjava.jar:./dirprm
```

10.2.27.4.3 Sample MySQL Database Target

```
gg.handlerlist=jdbcwriter
gg.handler.jdbcwriter.type=jdbc

#Handler properties for MySQL database target
gg.handler.jdbcwriter.DriverClass=com.mysql.jdbc.Driver
gg.handler.jdbcwriter.connectionURL=jdbc:<a target="_blank">
```

```

href="mysql://">mysql://</a><DBServer address>:3306/<database name>
gg.handler.jdbcwriter.userName=<dbuser>
gg.handler.jdbcwriter.password=<dbpassword>
gg.classpath=/path/to/mysql/jdbc/driver//mysql-connector-java-5.1.39-bin.jar

goldengate.userexit.timestamp=utc
goldengate.userexit.writers=javawriter
javawriter.stats.display=TRUE
javawriter.stats.full=TRUE
gg.log=log4j
gg.log.level=INFO
gg.report.time=30sec
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/
ggjava.jar:./dirprm

```

10.2.27.4.4 Sample MySQL Database Target with JDBC Metadata Provider

```

gg.handlerlist=jdbcwriter
gg.handler.jdbcwriter.type=jdbc

#Handler properties for MySQL database target with JDBC Metadata provider
gg.handler.jdbcwriter.DriverClass=com.mysql.jdbc.Driver
gg.handler.jdbcwriter.connectionURL=jdbc:mysql://<DBServer address>:3306/
<database name>
gg.handler.jdbcwriter.userName=<dbuser>
gg.handler.jdbcwriter.password=<dbpassword>
gg.classpath=/path/to/mysql/jdbc/driver//mysql-connector-java-5.1.39-bin.jar
#JDBC Metadata provider for MySQL target
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:mysql://<DBServer address>:3306/<database name>
gg.mdp.DriverClassName=com.mysql.jdbc.Driver
gg.mdp.UserName=<dbuser>
gg.mdp.Password=<dbpassword>

goldengate.userexit.timestamp=utc
goldengate.userexit.writers=javawriter
javawriter.stats.display=TRUE
javawriter.stats.full=TRUE
gg.log=log4j
gg.log.level=INFO
gg.report.time=30sec
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/
ggjava.jar:./dirprm

```

10.2.28 Microsoft Fabric Eventstreams

This chapter describes how to use the Microsoft Fabric eventstreams.

- [Overview](#)
- [Prerequisites](#)

10.2.28.1 Overview

Microsoft Fabric eventstreams is a feature in Microsoft Fabric that captures, transforms, and routes real-time events to various destinations. You can add event data sources, destinations, and transformations to the eventstream.

Microsoft Fabric eventstreams enables event ingestion without writing custom code. You can create an eventstream, add event data sources, and route to supported destinations.

Microsoft Fabric eventstreams exposes a Kafka-compatible endpoint that allows the Oracle GoldenGate Replicat to publish change data capture (CDC) events using the standard Apache Kafka protocol. This enables Oracle GoldenGate to stream transactional changes from supported sources directly into Fabric real-time intelligence pipelines without custom code.

Oracle GoldenGate uses the Kafka Handler to publish events to the Kafka endpoint exposed by Microsoft Fabric eventstreams.

10.2.28.2 Prerequisites

Before configuring Oracle GoldenGate for Microsoft Fabric eventstream:

1. Create a Microsoft Fabric eventstream with a custom endpoint source.
2. Retrieve the Kafka bootstrap servers, SASL credentials and Topic name from the custom endpoint (under SAS Key Authentication).
3. Ensure network accessibility from the GoldenGate deployment to the eventstream Kafka endpoint.

Following is a sample producer .properties to consume from Microsoft Fabric eventstream:

```
bootstrap.servers={YOUR_BOOTSTRAP_SERVERS}
key.serializer=org.apache.kafka.common.serialization.StringDeserializer
value.serializer=org.apache.kafka.common.serialization.StringDeserializer
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule
required.username="$ConnectionString" password="{YOUR_CONNECTION_STRING}";
```

Note

Ensure credentials and bootstrap server settings are correctly fetched from the Microsoft Fabric eventstream UI or API.

Related Topics

- [Microsoft Fabric eventstreams](#)

10.2.29 Microsoft Fabric OneLake

Microsoft Fabric is an end-to-end analytics and data platform designed for enterprises that require a unified solution. OneLake is built into the Fabric platform and provides a unified location to store all organizational data where the workloads operate. See <https://learn.microsoft.com/en-us/fabric/onelake/>. You can use the OneLake Event handler to load files that contain operations records into the following targets:

- Lakehouse in Microsoft Fabric

- Mirrored database in Microsoft Fabric

This topic contains the following:

- [OneLake Event Handler Prerequisites](#)
- [OneLake Mappings to Azure Data Lake Gen2](#)
- [OneLake Event Handler Configuration](#)
- [OneLake Event Handler Primary Key Update](#)
- [Ingesting into Fabric Mirrored Tables using Partitioning](#)
- [Managing the partnerEvents.json File for Fabric Mirrored Databases](#)
- [OneLake Event Handler Troubleshooting and Diagnostics](#)

10.2.29.1 OneLake Event Handler Prerequisites

- Azure cloud account set up.
- Microsoft Fabric set up.
 - Microsoft Fabric capacity along with workspace should exist.
 - Microsoft Fabric Lakehouse or Mirrored database should exist for the lakehouse or mirrored database target respectively.
 - Create a Microsoft Entra ID app to access the Microsoft Fabric workspace.
 - App needs to be granted at least the contributor role on the workspace.
 - Enable the app registration (service principal) to access Fabric APIs.
 - * **Admin Portal -> Tenant Settings -> Service principals can use Fabric APIs -> Enabled for the entire organization**
 - Enable remote access to data stored in OneLake
 - * **Admin Portal -> User can access data stored in OneLake using Apps external to Fabric.**
- Java Software Development Kit (SDK) for Azure Storage File Data Lake.

10.2.29.2 OneLake Mappings to Azure Data Lake Gen2

- **Storage Account:** An Azure storage account contains all of your Azure Storage data objects: blobs, file shares, queues, tables, and disks.
 - OneLake Storage Account name is always onelake.
- **Container:** A container organizes a set of blobs, similar to a directory in a file system. A storage account can include an unlimited number of containers, and a container can store an unlimited number of blobs.
 - OneLake container name is mapped to OneLake workspace name.
- **Endpoint:** The Azure Storage service endpoint.
 - OneLake default endpoint is <https://onelake.dfs.fabric.microsoft.com>, this can be overridden.

10.2.29.3 OneLake Event Handler Configuration

- [OneLake Event Handler Automatic Configuration](#)
- [File Writer Handler Configuration](#)
- [Autoconfiguration of Parquet/ORC Event Handler](#)
- [OneLake Event Handler Configuration](#)
- [File Format for the Lakehouse target](#)
- [OneLake Event Handler Classpath Configuration](#)
- [OneLake Event Handler Authentication](#)
- [OneLake Event Handler Proxy Configuration](#)
- [Sample Configuration for Lakehouse Target](#)
- [Sample Configuration for Mirrored Database Target](#)
- [Performance Considerations](#)

10.2.29.3.1 OneLake Event Handler Automatic Configuration

OneLake replication involves configuring multiple components, such as the File Writer Handler, Avro formatter, Parquet Event Handler, ORC Event Handler, and the OneLake Event Handler. The Automatic Configuration functionality will autoconfigure these components so that the user configuration is minimal. The properties modified by auto configuration would be logged in the handler log file.

To enable autoconfiguration to replicate data to the Lakehouse target, set the parameter `gg.target=fabric_lakehouse`.

To enable autoconfiguration to replicate data to the mirrored database target, set the parameter `gg.target=fabric_mirrored_database`.

10.2.29.3.2 File Writer Handler Configuration

The File Writer Handler name is pre set based on the `gg.target` configuration. For example, if `gg.target=fabric_lakehouse`, then the File Writer Handler name is set to the value `fabric_lakehouse` and its properties are automatically set to the required values for Lakehouse. As per this example, you can add or edit a property of the File Writer Handler as follows: `gg.handler.fabric_lakehouse.inactivityRollInterval=1m`.

10.2.29.3.3 Autoconfiguration of Parquet/ORC Event Handler

Event Handler name is pre-set to the value `parquet` or `orc` based on the file format configuration.

- [OneLake Event Handler File Format Configuration for Parquet/ORC](#)

10.2.29.3.3.1 OneLake Event Handler File Format Configuration for Parquet/ORC

- For use cases that require Parquet files such as Open Mirroring and vanilla Parquet format, Autoconfiguration will configure the Avro formatter and chains it with a Parquet event handler, and the OneLake event handler. This is configured as follows: `gg.format=parquet`

Note

For the Open Mirroring target (`gg.target=fabric_mirrored_database`), the file format configuration is internal and cannot be modified.

- For use case that requires ORC files, Autoconfiguration will configure the Avro formatter and chains it with the ORC event handler, and the OneLake event handler. This is configured as follows: `gg.format=orc`.

10.2.29.3.4 OneLake Event Handler Configuration

OneLake Event Handler name is pre set to the value `onelake`.

`gg.target` must be set to one of the following values:

- `fabric_lakehouse`: To replicate to Lakehouse in Microsoft Fabric.
- `fabric_mirrored_database`: To replicate to Mirrored Database in Microsoft Fabric.

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.onelake.workspace</code>	Required	String	None	Sets the Microsoft Fabric workspace name.
<code>gg.eventhandler.onelake.lakehouse</code>	Required	String	None	Applicable only to the Lakehouse target. Sets the Microsoft Fabric lakehouse name.
<code>gg.eventhandler.onelake.mirror</code>	Required	String	None	Applicable only to the mirrored database target. Sets the mirrored database name in Fabric.
<code>gg.eventhandler.onelake.tenantId</code>	Optional	String	None	Sets the Azure tenant ID of the application.
<code>gg.eventhandler.onelake.clientId</code>	Optional	String	None	Sets the Azure client ID of the application.
<code>gg.eventhandler.onelake.clientSecret</code>	Optional	String	None	Sets the Azure client secret for the authentication.

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .onelake.pathMappingTemplate	Optional	A string with resolvable keywords and constants used to dynamically generate the landing path for data files into OneLake.	If gg.target is set to fabric_mirrored_database, then the default value is \$ {catalogname}.MountedRelationalDatabase/Files/LandingZone/\$ {schemaname}.schema/\$ {tablename}. This cannot be modified. If gg.target=fabric_lakehouse, then the default value is \$ {catalogname}.lakehouse/Files/ogg/\$ {groupName}/\$ {schemaname}.schema/\$ {tablename}, this can be modified.	Use keywords interlaced with constants to dynamically generate a path names at runtime. Example path name would be: ogg/data/\$ {fullyQualifiedTableName}. For more information about the supported keywords, see Template Keywords .
gg.eventhandler .onelake.fileNameMappingTemplate	Optional	A string with resolvable keywords and constants used to dynamically generate the data file names at runtime.	If gg.format is set to fabric_mirrored_database, then this value is set to \$ {custom[]} and cannot be edited. If gg.target=fabric_lakehouse, then the default value is based on the upstream handler, and can be modified.	Use keywords interlaced with constants to dynamically generate a unique file name at runtime. Typically, file names follow the format, \$ {fullyQualifiedTableName}_\$ {groupName}_\$ {currentTimestamp}.txt.
gg.eventhandler .onelake.endpoint	Optional	String	https://onelake.dfs.fabric.microsoft.com	Sets the Fabric OneLake endpoint.
gg.format	Optional	parquet, orc, or one of the GG for DAA pluggable formatter name.	parquet	Applicable only to the Lakehouse target. Sets the Fabric OneLake file format. For more information, see File Format for the Lakehouse target .

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler. onelake.replicatGroupPartitio nName	Optional	String	ogg_group_name	Applicable only if gg.target is set to fabric_mirrored_database. A string to specify the Fabric partition column name. Value should be a string between 1 and 20 characters in length.
gg.eventhandler. onelake.maxRet ries	Optional	Integer	3	The maximum number of retries to attempt if the file upload to OneLake fails. Value should be between 1 and 20.
gg.eventhandler. onelake.maxRet ries	Optional	Integer	10	The initial delay between retries (in seconds). The retry delay is initially set to the specified value and doubles with each retry. Value should be between 1 and 120.
gg.keyupdate.th readrange.behav ior	Optional	ABEND, WARN	If gg.target is set to fabric_mirrored_database, then the default value is ABEND. For all the other targets, the default value is not set.	Controls Replicat's behavior when it encounters a key update operation for a table configured with THREADRANGE clause. ABEND aborts processing, while WARN logs a warning.

10.2.29.3.5 File Format for the Lakehouse target

The parameter gg.format can be configured to set the file format.

It can be set to one of the following values:

- parquet: Generate Parquet format files.
- orc: Generate ORC format files.
- Any other pluggable format supported by Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

10.2.29.3.6 OneLake Event Handler Classpath Configuration

Ensure that the classpath includes the path to the following dependencies:

- Parquet Event handler dependencies including Hadoop dependencies.
- Azure Storage File DataLake Java SDK.

Ensure that the classpath includes the path to the following dependencies in the given order:

1. OneLake Event Handler dependencies.
 2. Parquet Event Handler dependencies, including Hadoop dependencies.
 3. Azure Storage File DataLake Java SDK.
- [OneLake Event Handler Dependencies](#)

10.2.29.3.6.1 OneLake Event Handler Dependencies

The dependency downloader script `oneLake.sh` can be used to download the OneLake dependencies. Alternatively, you can manually download the OneLake dependencies using the following maven co-ordinates:

```
<dependencies>
  <dependency>
    <groupId>com.azure</groupId>
    <artifactId>azure-storage-file-datalake</artifactId>
    <version>12.20.0</version>
  </dependency>
  <dependency>
    <groupId>com.azure</groupId>
    <artifactId>azure-identity</artifactId>
    <version>1.13.1</version>
  </dependency>
</dependencies>
```

Edit the `gg.classpath` configuration parameter to include the path to the Azure Storage File Data Lake SDK.

10.2.29.3.7 OneLake Event Handler Authentication

You can authenticate the Azure Storage device by configuring following:

- tenantID
- clientId
- clientSecret
- [Azure Tenant ID, Client ID, and Client Secret](#)

10.2.29.3.7.1 Azure Tenant ID, Client ID, and Client Secret

To obtain your Azure tenant ID:

- Go to the Microsoft Azure portal.
- Select Azure Active Directory from the list on the left to view the Azure Active Directory panel.

- Select Properties in the Azure Active Directory panel to view the Azure Active Directory properties.
The Azure tenant ID is the field marked as Directory ID.
- To obtain your Azure client ID and client secret:
 - Go to the Microsoft Azure portal.
 - Select **All Services** from the list on the left to view the Azure Services Listing.
 - Type **App** into the filter command box and select **App Registrations** from the listed services.
 - Select the App Registration that you have created to access Microsoft Fabric workspace.
The Application id displayed for the App Registration is the client ID. The client secret is the generated key string when a new key is added.

This generated key string is available only once when the key is created. If you do not know the generated key string, then create another key making sure you capture the generated key string.

10.2.29.3.8 OneLake Event Handler Proxy Configuration

When the process is run behind a proxy server, the property `jvm.bootoptions` can be used to set proxy server configuration using well-known Java proxy properties. For example:

```
jvm.bootoptions=-Dhttps.proxyHost=some-proxy-address.com -Dhttps.proxyPort=80
-Djava.net.useSystemProxies=true
```

10.2.29.3.9 Sample Configuration for Lakehouse Target

```
gg.target=fabric_lakehouse
#TODO: format can be 'parquet' or 'orc' or one of the pluggable formatter types. Default
is 'parquet'.
#gg.format=parquet
#TODO: Edit the Fabric workspace name.
gg.eventhandler.onelake.workspace=<workspace-name>
#TODO: Edit the Fabric lakehouse name.
gg.eventhandler.onelake.lakehouse=<lakehouse-name>
#TODO: Edit the tenant ID of the application.
gg.eventhandler.onelake.tenantId=<azure-tenant-id>
#TODO: Edit the client ID of the application.
gg.eventhandler.onelake.clientId=<azure-client-id>
#TODO: Edit the client secret for the authentication.
gg.eventhandler.onelake.clientSecret=<azure-client-secret>
#TODO: Edit the classpath to include Hadoop, Parquet, and Azure DataLake SDK
dependencies.
gg.classpath=$THIRD_PARTY_DIR/onelake/*:$THIRD_PARTY_DIR/hadoop/*:$THIRD_PARTY_DIR/
parquet/
#TODO: Edit the proxy configuration.
#jvm.bootoptions=-Dhttps.proxyHost=some-proxy-address.com -Dhttps.proxyPort=80 -
Djava.net.useSystemProxies=true
```

10.2.29.3.10 Sample Configuration for Mirrored Database Target

```
gg.target=fabric_mirrored_database
#TODO: Edit the Fabric workspace name.
gg.eventhandler.onelake.workspace=<workspace-name>
```

```
#TODO: Edit the Fabric mirror Database name.
gg.eventhandler.onelake.mirror=<mirror-name>
#TODO: Edit the tenant ID of the application.
gg.eventhandler.onelake.tenantId=<azure-tenant-id>
#TODO: Edit the client ID of the application.
gg.eventhandler.onelake.clientId=<azure-client-id>
#TODO: Edit the client secret for the authentication.
gg.eventhandler.onelake.clientSecret=<azure-client-secret>#TODO: Edit the classpath to
include Hadoop, Parquet, and Azure DataLake SDK dependencies.
gg.classpath=$THIRD_PARTY_DIR/hadoop/*:$THIRD_PARTY_DIR/parquet/*:$THIRD_PARTY_DIR/
onelake/*
#TODO: Edit the proxy configuration.
#jvm.bootoptions=-Dhttps.proxyHost=some-proxy-address.com -Dhttps.proxyPort=80 -
Djava.net.useSystemProxies=true
```

10.2.29.3.11 Performance Considerations

You can set GROUPTRANSOPS to a higher value (recommended up to 20000). By default, the value is set to 1000.

For initial load replicats, you can create multiple replicats with the classic mode and configure each replicat to handle one individual table or a discrete subset of tables.

For real-time replicats, you can use coordinated replicat mode to improve performance. While configuring coordinated replicat, tables should be mapped to individual threads.

10.2.29.4 OneLake Event Handler Primary Key Update

Primary key UPDATE behavior depends on the file-format configuration.

- [Mirrored Database in Microsoft Fabric](#)
- [Microsoft Fabric OneLake Initial Load](#)
- [Lakehouse in Microsoft Fabric](#)

10.2.29.4.1 Mirrored Database in Microsoft Fabric

When file format is set to `gg.format=fabric_mirroring`, then primary key UPDATE operations will be split into a DELETE operation followed by an INSERT operation. This behavior cannot be modified.

10.2.29.4.2 Microsoft Fabric OneLake Initial Load

Transitioning from Initial Load to CDC Replication

You can follow the provided sequence to safely transition from Initial Load replication to Change Data Capture (CDC) replication.

1. Run the **Initial Load Replicat**.
2. Allow the **Initial Load Replicat** to complete loading all table data into the target OneLake location.
3. Validate Initial Load completion:
 - Verify that all expected records are successfully replicated to the target.
 - Confirm row counts and data consistency between the source and target before you proceed.

4. Stop and remove the **Initial Load Replicat** after successful validation, if it is no longer required.
5. Delete the OneLake scratch pad directories:
 - Delete the scratchPad sub directory for all tables loaded through the Initial Load Replicat.
 - You can delete the scratchPad directory by using Azure Storage Explorer or REST API commands.
6. Start the CDC Replicat.
7. Start the CDC Replicat (CDREP1) only after the scratch pad cleanup is complete. CDC replication should then successfully acquire ownership of the table directory.

10.2.29.4.3 Lakehouse in Microsoft Fabric

If `gg.target=fabric_lakehouse` is set, then by default primary key UPDATE operations will result in a Replicat ABEND.

This behavior can be modified by configuration of the formatter property `gg.handler.onelake.format.pkUpdateHandling`

The property `gg.handler.onelake.format.pkUpdateHandling` can accept one of the following input:

- `abend`: ABEND replicat when a primary key UPDATE is processed.
- `update`: Replicat processes primary key UPDATE as a regular UPDATE.
- `delete-insert`: Replicat would split primary key UPDATE into a DELETE operation followed by an INSERT operation.

10.2.29.5 Ingesting into Fabric Mirrored Tables using Partitioning

The existing solution for ingesting data into Fabric mirrored tables uses a single landing folder for a specific table and a metadata file to specify key columns. However, this approach does not scale well for large tables as only a single replicat thread is used. To improve scalability, we can leverage Fabric's partitioning feature and Oracle GoldenGate's coordinated apply feature to distribute the workload for specific tables.

- [Replicat MAP Statement](#)
- [Metadata File Updates](#)
- [Limitation](#)

10.2.29.5.1 Replicat MAP Statement

For tables that require partitioning, the Oracle GoldenGate coordinated apply feature is used to distribute the workload. For example:

```
MAP scott.orders, TARGET scott.orders, THREADRANGE (1-2, 0ID);
```

The data files are stored into the following folders:

```
${catalogname}.MountedRelationalDatabase/Files/LandingZone/${schemaname}.schema/${tablename}/ogg_group_name=REP001/
${catalogname}.MountedRelationalDatabase/Files/LandingZone/${schemaname}.schema/${tablename}/ogg_group_name=REP002/
```

- The partition column `ogg_group_name` is populated with the respective replicat group name (group name of the coordinated worker thread).
- The additional column `ogg_group_name` is visible for every user table on the Fabric Mirrored database.
- Customers can configure `THREADRANGE` as required.

10.2.29.5.2 Metadata File Updates

For tables configured with `THREADRANGE`, the Fabric metadata file is updated as follows:

```
{ "keyColumns": ["ogg_group_name", "KEY_1", "KEY_2"], "IsPartitioned": "true"
}
```

The metadata file is placed in the base table folder as:

```
${catalogname}.MountedRelationalDatabase/Files/LandingZone/${schemaname}.schema/${tablename}.
```

For tables that do not require partitioning, coordinated apply or `THREADRANGE` configuration is not required.

The Fabric metadata file for non-partitioned tables is as follows:

```
{ "keyColumns": ["KEY_1", "KEY_2"], "IsPartitioned": "false" }
```

10.2.29.5.3 Limitation

When partitioning is enabled for a table, that is, a table configured with `THREADRANGE` clause, in-place primary key update operations may be applied out of order.

To manage this behavior, use the `gg.keyupdate.threadrange.behavior` parameter, which allows you to control how Replicat handles such operations.

10.2.29.6 Managing the `partnerEvents.json` File for Fabric Mirrored Databases

Oracle GoldenGate generates a `partnerEvents.json` file in the landing zone when you configure a Fabric Mirrored Database with partner event tagging enabled.

① Note

Oracle recommends implementing this file as part of the standard Oracle GoldenGate integration, although Microsoft considers it optional.

File Location

Oracle GoldenGate writes the `partnerEvents.json` file at the mirrored database level in the landing zone, rather than at the individual table level.

Behavior

- Oracle GoldenGate creates and maintains the `partnerEvents.json` file during Replicat initialization.
- Changes to the source configuration are automatically reflected in the file.
- The file is lightweight and does not affect replication performance.

Considerations

- Oracle GoldenGate manages the file as part of the Fabric Open Mirroring integration.
- The file improves visibility and troubleshooting capabilities within Microsoft Fabric.
- The file does not introduce any performance overhead on replication processes.

10.2.29.7 OneLake Event Handler Troubleshooting and Diagnostics

- **Unsupported Operations:**
 - DDL operations that DROP/RENAME table will not be replicated by the Replicat process.
 - Renaming columns of the table is not supported by the Microsoft application consuming the Fabric Mirroring format file.
 - TRUNCATE operations cannot be replicated.

- **Error:**

```
com.azure.identity.CredentialUnavailableException: EnvironmentCredential authentication unavailable.Environment variables are not fully configured.
```

This indicates that the Azure authentication parameters `tenantId`, `clientId`, and `clientSecret` are not configured. See [Azure Tenant ID, Client ID, and Client Secret](#) to configure the authentication parameters.

- **Error:**

```
java.lang.IllegalArgumentException: Invalid tenant id provided. You can locate your tenant id by following the instructions listed here: https://learn.microsoft.com/partner-center/find-ids-and-domain-names
```

This indicates that the authentication parameter `tenandId` is invalid. See [Azure Tenant ID, Client ID, and Client Secret](#) to configure the authentication parameters.

- **Error:**

```
com.microsoft.aad.msal4j.MsalServiceException: AADSTS700016: Application with identifier '<invalid_clientId>' was not found in the directory '<tenant name>'.

```


This indicates that the authentication parameter `clientId` with value `<invalid_client_id>` is incorrect. See [Azure Tenant ID, Client ID, and Client Secret](#) to configure the authentication parameters.

- **Error:**

```
com.microsoft.aad.msal4j.MsalServiceException: AADSTS7000215: Invalid
client secret provided.
```

This indicates that the authentication parameter `clientSecret` is incorrect. See [Azure Tenant ID, Client ID, and Client Secret](#) to configure the authentication parameters.

- **Error:**

```
com.azure.storage.file.datalake.models.DataLakeStorageException: Status
code 404,
{"error":{"code":"ArtifactNotFound","message":"Request Failed with
Artifact 'gglakehouse1_invalid.lakehouse' is not found in workspace
'ggworkspace1'."}}
```

This indicates that the Fabric workspace name or lakehouse name is invalid. If the Fabric workspace or lakehouse does not exist, then you should create these before starting the replicat process. Ensure the configuration parameters `gg.eventhandler.onelake.workspace` and `gg.eventhandler.onelake.lakehouse` are set to the Fabric warehouse and lakehouse names respectively.

- **Error:**

```
ONELAKE-00073 The event handler cannot proceed. The stage file
'<file_name>' in the directory '<directory_name>' contains one or more
truncate operations.
Truncate operations cannot be replicated into Microsoft Fabric OneLake
Generic Mirror. Modify the GoldenGate replicat parameter file and remove
the line that contains GETTRUNCATES and
restart the replicat process.
```

There are one or more TRUNCATE operations that were processed by the replicat process. To proceed, you need to remove the GETTRUNCATES parameter from the parameter file and restart the replicat process.

- **Error:**

```
ONELAKE-00082 File name sequence number for table QASOURCE.TCUSTMER has
reached the maximum limit of 99,999,999,999,999,999.
```

You need to clear the backlog in OneLake or purge the last file with the highest sequence number and restart the replicat process.

- **Error:**

```
The operation record in the trail sequence'<seqno>' at offset '<offset>'
for the table '<table>' has missing column values.
```

OneLake replication requires full images. You need to regenerate the trail files that contain full images for UPDATE operations, and restart the replication process.

- **Mirrored Database Target:**

- **Error:**

ONELAKE-00073 The event handler cannot proceed. The stage file '<file_name>' in the directory '<directory_name>' contains one or more truncate operations. Truncate operations cannot be replicated into Mirrored Database in Microsoft Fabric. Modify the GoldenGate replicat parameter file and remove the line that contains GETTRUNCATES and restart the replicat process.

There are one or more TRUNCATE operations that were processed by the replicat process.

To proceed, you need to remove the GETTRUNCATES parameter from the parameter file and restart the replicat process.

- **Error:**

ONELAKE-00082 File name sequence number for table QASOURCE.TCUSTMER has reached the maximum limit of 99,999,999,999,999,999,999.

User needs to clear the backlog in OneLake or purge the last file with the highest sequence number and restart the replicat process.

- **Error:**

The operation record in the trail sequence '<seqno>' at offset '<offset>' for the table '<table>' has missing column values

Replication to Mirrored Database in Microsoft Fabric requires full images.

You need to regenerate the trail files that contain full images for UPDATE operations, and restart the replication process.

- **Error:**

```
Azure Identity => ERROR in getToken() call for scopes [https://
storage.azure.com/.default]: java.io.UncheckedIOException:
io.netty.channel.StacklessClosedChannelException
{"az.sdk.message":"Failed to acquire a new access
token.", "exception":"java.io.UncheckedIOException:
io.netty.channel.StacklessClosedChannelException"}
```

The root cause for this issue is the definition order of the downloaded dependencies in gg.classpath in the replicat properties file. The correct order is as follows:

```
gg.classpath=$THIRD_PARTY_DIR/onelake/*:$THIRD_PARTY_DIR/hadoop/
*:$THIRD_PARTY_DIR/parquet/*
```

- The current replicat group CDREP1 has a valid checkpoint set by the replicat process and there is another replicat group ILREP1 writing files to the directory RDLSPO2A.MountedRelationalDatabase/Files/LandingZone/INVESTRV.schema/DISB_FUND_INV_SUMMARY for table INVESTRV.DISB_FUND_INV_SUMMARY.

Only one replicat group can write files to a specific table directory on OneLake. This issue occurs when both the Initial Load replicat and CDC replicat attempt to write to the same OneLake table directory simultaneously. OneLake allows only a single replicat group to own and write to a specific target table directory at a time. .

ONELAKE-00092 - Invalid State During CDC Replicat Startup

ONELAKE-00092 Invalid state.

Follow the sequence of steps mentioned under safely transition from *Initial Load* to *CDC replication*.

10.2.30 MongoDB

Learn how to use the MongoDB Handler, which can replicate transactional data from Oracle GoldenGate to a target MongoDB and Autonomous JSON databases (AJD and ATP) .

- [Overview](#)
- [MongoDB Wire Protocol](#)
- [Supported Target Types](#)
- [Detailed Functionality](#)
- [Setting Up and Running the MongoDB Handler](#)
- [Security and Authentication](#)
- [Reviewing Sample Configurations](#)
- [MongoDB to AJD/ATP Migration](#)
- [Configuring an Initial Synchronization of Extract for a MongoDB Source Database using Precise Instantiation](#)
- [Delivery to Oracle AI Database JSON Collection Tables](#)
- [MongoDB Handler Client Dependencies](#)

What are the dependencies for the MongoDB Handler to connect to MongoDB databases?

10.2.30.1 Overview

Mongodb Handler can be used to replicate data from RDMS as well as document based databases like MongoDB or Cassandra to the following target databases using MongoDB wire protocol

10.2.30.2 MongoDB Wire Protocol

The MongoDB Wire Protocol is a simple socket-based, request-response style protocol. Clients communicate with the database server through a regular TCP/IP socket, see <https://docs.mongodb.com/manual/reference/mongodb-wire-protocol/>.

10.2.30.3 Supported Target Types

- MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling, see <https://www.mongodb.com/>.

- Oracle Autonomous JSON Database (AJD) is a cloud document database service that makes it simple to develop JSON-centric applications, see [Autonomous JSON Database | Oracle](#).
- Autonomous Database for transaction processing and mixed workloads (ATP) is a fully automated database service optimized to run transactional, analytical, and batch workloads concurrently, see [Autonomous Transaction Processing | Oracle](#).
- On-premises Oracle Database 21c with Database API for MongoDB is also a supported target. See [Installing Database API for MongoDB for any Oracle Database](#).

10.2.30.4 Detailed Functionality

The MongoDB Handler takes operations from the source trail file and creates corresponding documents in the target MongoDB or Autonomous databases (AJD and ATP).

A record in MongoDB is a Binary JSON (BSON) document, which is a data structure composed of field and value pairs. A BSON data structure is a binary representation of JSON documents. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

A collection is a grouping of MongoDB or AJD/ATP documents and is the equivalent of an RDBMS table. In MongoDB or AJD/ATP databases, a collection holds collection of documents. Collections do not enforce a schema. MongoDB or AJD/ATP documents within a collection can have different fields.

- [Document Key Column](#)
- [Primary Key Update Operation](#)
- [MongoDB Trail Data Types](#)

10.2.30.4.1 Document Key Column

MongoDB or AJD/ATP databases require every document (row) to have a column named `_id` whose value should be unique in a collection (table). This is similar to a primary key for RDBMS tables. If a document does not contain a top-level `_id` column during an insert, the MongoDB driver adds this column.

The MongoDB Handler builds custom `_id` field values for every document based on the primary key column values in the trail record. This custom `_id` is built using all the key column values concatenated by a `:` (colon) separator. For example:

```
KeyColValue1:KeyColValue2:KeyColValue3
```

The MongoDB Handler enforces uniqueness based on these custom `_id` values. This means that every record in the trail must be unique based on the primary key columns values. Existence of non-unique records for the same table results in a MongoDB Handler failure and in Replicat abending with a duplicate key error.

The behavior of the `_id` field is:

- By default, MongoDB creates a unique index on the column during the creation of a collection.
- It is always the first column in a document.
- It may contain values of any BSON data type except an array.

10.2.30.4.2 Primary Key Update Operation

MongoDB or AJD/ATP databases do not allow the `_id` column to be modified. This means a primary key update operation record in the trail needs special handling. The MongoDB Handler converts a primary key update operation into a combination of a DELETE (with old key) and an INSERT (with new key). To perform the INSERT, a complete before-image of the update operation in trail is recommended. You can generate the trail to populate a complete before image for update operations by enabling the Oracle GoldenGate GETUPDATEBEFORES and NOCOMPRESSUPDATES parameters.

10.2.30.4.3 MongoDB Trail Data Types

The MongoDB Handler supports delivery to the BSON data types as follows:

- 32-bit integer
- 64-bit integer
- Double
- Date
- String
- Binary data

10.2.30.5 Setting Up and Running the MongoDB Handler

The following topics provide instructions for configuring the MongoDB Handler components and running the handler.

- [Classpath Configuration](#)
- [MongoDB Handler Configuration](#)
- [Using Bulk Write](#)
- [Using Write Concern](#)
- [Using Three-Part Table Names](#)
- [Using Undo Handling](#)

10.2.30.5.1 Classpath Configuration

The MongoDB Java Driver is required for Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) to connect and stream data to MongoDB. If the GG for DAA version is 21.7.0.0.0 and below, then you need to use 3.x ([MongoDB Java Driver 3.12.8](#)). If the GG for DAA version is 21.8.0.0.0 and above, then you need to use [MongoDB Java Driver 4.6.0](#). The MongoDB Java Driver is not included in the GG for DAA product. You must download the driver from: [mongo java driver](#).

Select **mongo-java-driver** and the version to download the recommended driver JAR file.

You must configure the `gg.classpath` variable to load the MongoDB Java Driver JAR at runtime. For example: `gg.classpath=/home/mongodb/mongo-java-driver-3.12.8.jar`.

GG for DAA supports the MongoDB Decimal 128 data type that was added in MongoDB 3.4. Use of a MongoDB Java Driver prior to 3.12.8 results in a `ClassNotFoundException`.

10.2.30.5.2 MongoDB Handler Configuration

You configure the MongoDB Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the MongoDB Handler, you must first configure the handler type by specifying `gg.handler.name.type=mongodb` and the other MongoDB properties as follows:

Table 10-35 MongoDB Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.type</code>	Required	<code>mongodb</code>	None	Selects the MongoDB Handler for use with Replicat.
<code>gg.handler.name.bulkwrite</code>	Optional	<code>true</code> <code>false</code>	<code>true</code>	Set to <code>true</code> , the handler caches operations until a commit transaction event is received. When committing the transaction event, all the cached operations are written out to the target MongoDB, AJD and ATP databases, which provides improved throughput. Set to <code>false</code> , there is no caching within the handler and operations are immediately written to the MongoDB, AJD and ATP databases.
<code>gg.handler.name.writeConcern</code>	Optional	<code>{"w": "value", "wtimeout": "number"}</code>	None	Sets the required write concern for all the operations performed by the MongoDB Handler. The property value is in JSON format and can only accept keys as <code>w</code> and <code>wtimeout</code> , see https://docs.name.com/manual/reference/write-concern/ .
<code>gg.handler.name.clientURI</code>	Optional	Valid MongoDB client URI	None	Sets the MongoDB client URI. A client URI can also be used to set other MongoDB connection properties, such as authentication and WriteConcern.
<code>gg.handler.name.checkMaxRowSizeLimit</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	When set to <code>true</code> , the handler verifies that the size of the BSON document inserted or modified is within the limits defined by the MongoDB database. Calculating the size involves the use of a default codec to generate a <code>RawBsonDocument</code> , leading to a small degradation in the throughput of the MongoDB Handler. If the size of the document exceeds the MongoDB limit, an exception occurs and Replicat abends.
<code>gg.handler.name.upsert</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	Set to <code>true</code> , a new Mongo document is inserted if there are no matches to the query filter when performing an UPDATE operation.

Table 10-35 (Cont.) MongoDB Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.enableDecimal128</code>	Optional	<code>true</code> <code>false</code>	<code>true</code>	MongoDB version 3.4 added support for a 128-bit decimal data type called <code>Decimal128</code> . This data type was needed since Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) supports both integer and decimal data types that do not fit into a 64-bit Long or Double. Setting this property to <code>true</code> enables mapping into the <code>Double128</code> data type for source data types that require it. Set to <code>false</code> to process these source data types as 64-bit Doubles.
<code>gg.handler.name.enableTransactions</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	Set to <code>true</code> , to enable transactional processing in MongoDB 4.0 and higher.

Note

MongoDB added support for transactions in MongoDB version 4.0. Additionally, the minimum version of the MongoDB client driver is 3.10.1.

10.2.30.5.3 Using Bulk Write

Bulk write is enabled by default. For better throughput, Oracle recommends that you use bulk write.

You can also enable bulk write by using the `BulkWrite` handler property. To enable or disable bulk write use the `gg.handler.handler.BulkWrite=true | false`. The MongoDB Handler does *not* use the `gg.handler.handler.mode=op | tx` property that is used by Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

With bulk write, the MongoDB Handler uses the `GROUPTRANSOPS` parameter to retrieve the batch size. The handler converts a batch of trail records to MongoDB documents, which are then written to the database in one request.

10.2.30.5.4 Using Write Concern

Write concern describes the level of acknowledgement that is requested from MongoDB for write operations to a standalone MongoDB, replica sets, and sharded-clusters. With sharded-clusters, Mongo instances pass the write concern on to the shards, see <https://docs.mongodb.com/manual/reference/write-concern/>.

Use the following configuration:

`w: value`
`wtimeout: number`

10.2.30.5.5 Using Three-Part Table Names

An Oracle GoldenGate trail may have data for sources that support three-part table names, such as *Catalog.Schema.Table*. MongoDB only supports two-part names, such as *DBName.Collection*. To support the mapping of source three-part names to MongoDB two-part names, the source *Catalog* and *Schema* is concatenated with an underscore delimiter to construct the Mongo *DBName*.

For example, *Catalog.Schema.Table* would become *catalog1_schema1.table1*.

10.2.30.5.6 Using Undo Handling

The MongoDB Handler can recover from bulk write errors using a lightweight undo engine. This engine works differently from typical RDBMS undo engines, rather the best effort to assist you in error recovery. Error recovery works well when there are primary violations or any other bulk write error where the MongoDB database provides information about the point of failure through `BulkWriteException`.

[Table 10-36](#) Table 1 lists the requirements to make the best use of this functionality.

Table 10-36 Undo Handling Requirements

Operation to Undo	Require Full Before Image in the Trail?
INSERT	No
DELETE	Yes
UPDATE	No (before image of fields in the SET clause.)

If there are errors during undo operations, it may be not possible to get the MongoDB collections to a consistent state. In this case, you must manually reconcile the data.

10.2.30.6 Security and Authentication

MongoDB Handler uses connection URI to connect to a MongoDB deployment. Authentication and Security is passed as query string as part of connection URI.

- [MongoDB URI-Based Authentication](#)
- [TLS/SSL Configuration](#)
MongoDB connections can be secured using TLS or mutual TLS (mTLS).
- [SSL Configuration Setup](#)
- [MongoDB Extract and Replicat Configuration](#)
For secure connection to MongoDB cluster with X509 authentication and authorization enabled, the connection URI must include parameters for the authentication type, client certificate, and CA certificate. Also, we need to pass the client and server trust-store/key-store files and their passwords in JVM boot options.

10.2.30.6.1 MongoDB URI-Based Authentication

MongoDB uses a URI-style connection string that supports various authentication methods and options as query parameters.

- [Authentication Methods](#)

10.2.30.6.1.1 Authentication Methods

- **SCRAM-SHA-256** : Recommended password-based authentication mechanism.
Example: `mongodb://username@host:27017/?authSource=admin&authMechanism=SCRAM-SHA-256`
- **SCRAM-SHA-1** : For backward compatibility.
Example: `mongodb://username@host:27017/?authSource=admin&authMechanism=SCRAM-SHA-1`
- **PLAIN (LDAP)**: Requires TLS.
Example: `mongodb://ldapUser@host:27017/?authMechanism=PLAIN&authSource=$external&tls=true`
- **x.509 Certificate Authentication:**
Example: `mongodb://host:27017/?authMechanism=MONGODB-X509&tls=true&tlsCertificateKeyFile=client.pem&tlsCAFile=ca.pem`

① Note

For more information on X509 configuration steps, see [MongoDB Extract and Replicat Configuration](#).

10.2.30.6.2 TLS/SSL Configuration

MongoDB connections can be secured using TLS or mutual TLS (mTLS).

① Note

For more information, see [SSL Configuration Setup](#) to generate the certificates required for `tls`/`mtls`.

- [Disable TLS \(Not Recommended\)](#)
- [Enable TLS](#)
- [Mutual TLS \(mTLS\)](#)

10.2.30.6.2.1 Disable TLS (Not Recommended)

Example: `mongodb://host:27017/?tls=false`

10.2.30.6.2.2 Enable TLS

- If server uses the CA-trusted certificate:
Example: `mongodb://host:27017/?tls=true`
- If server uses the self-signed certificate:

Example: `mongodb://host:27017/?
tls=true&tlsAllowInvalidCertificates=true`

or

Example: `mongodb://host:27017/?tls=true`

Note

The truststore information should be provided in the JVM bootoptions of replicat properties as `-Djavax.net.ssl.trustStore=/path/to/truststore.jks`
`-Djavax.net.ssl.trustStorePassword=changeit`

10.2.30.6.2.3 Mutual TLS (mTLS)

Performs the client certificate validation along with the server certificate validation.

Example: `mongodb://host:27017/?tls=true`

Note

The truststore and keystore information should be provided in the JVM bootoptions. JVM bootoptions for mTLS:

```
-Djavax.net.ssl.trustStore=/path/to/truststore.jks -  
Djavax.net.ssl.trustStorePassword=changeit -  
Djavax.net.ssl.keyStore=/path/to/keystore.jks -  
Djavax.net.ssl.keyStorePassword=changeit
```

10.2.30.6.3 SSL Configuration Setup

To configure SSL between the MongoDB instance and Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) MongoDB Handler, do the following:

Create certificate authority (CA)

```
openssl req -passout pass:password -new -x509 -days 3650 -extensions v3_ca -  
keyout  
ca_private.pem -out ca.pem -subj  
"/CN=CA/OU=GOLDENGATE/O=ORACLE/L=BANGALORE/ST=KA/C=IN"
```

Create key and certificate signing requests (CSR) for client and all server nodes

```
openssl req -newkey rsa:4096 -nodes -out client.csr -keyout client.key -subj  
'/CN=certName/OU=OGGDBCLIENT/O=ORACLE/L=BANGALORE/ST=AP/C=IN'
```

```
openssl req -newkey rsa:4096 -nodes -out server.csr -keyout server.key -subj  
'/CN=slc13auo.us.oracle.com/OU=GOLDENGATE/O=ORACLE/L=BANGALORE/ST=TN/C=IN'
```

Sign the certificate signing requests with CA

```
openssl x509 -passin pass:password -sha256 -req -days 365 -in client.csr -CA
ca.pem -CAkey
ca_private.pem -CAcreateserial -out client-signed.crt

openssl x509 -passin pass:password -sha256 -req -days 365 -in server.csr -CA
ca.pem -CAkey
ca_private.pem -CAcreateserial -out server-signed.crt -extensions v3_req -
extfile
<(cat << EOF[ v3_req ]subjectAltName = @alt_names
[ alt_names ]
DNS.1 = 127.0.0.1
DNS.2 = localhost
DNS.3 = hostname
EOF)
```

Create the privacy enhanced mail (PEM) file for mongod

```
cat client-signed.crt client.key > client.pem
cat server-signed.crt server.key > server.pem
```

Create trust store and keystore

```
openssl pkcs12 -export -out server.pkcs12 -in server.pem
openssl pkcs12 -export -out client.pkcs12 -in client.pem
```

Note

Provide the password for server and client trust-store when prompted.

```
bash-4.2$ ls
ca.pem ca_private.pem client.csr client.pem server-signed.crt
server.key server.pkcs12
ca.srl client-signed.crt client.key client.pkcs12 server.csr
server.pem
```

Get client certificate subject for MongoDB username

```
openssl x509 -in <pathToClientPEM> -inform PEM -subject -nameopt RFC2253
```

Example Output:

```
subject= CN=myName,OU=myOrgUnit,O=myOrg,L=myLocality,ST=myState,C=myCountry
```

Start MongoDB instance without TLS/SSL to create a user

Start single node of MongoDB without authentication and connect with MongoDB shell to create a user with user name.

Note

The user name should match with the client certificate subject received for MongoDB and the role as admin.

```
mongosh --bind_ip <hostname> --port <port>

db.getSiblingDB("$external").runCommand({
  createUser: "
CN=myName,OU=myOrgUnit,O=myOrg,L=myLocality,ST=myState,C=myCountry ",
  roles: [{ role: "root", db: "admin" }]
});
```

Restart MongoDB cluster with TLS/SSL enabled

Start each instance of MongoDB cluster with mongod command passing the following options to enable SSL/TLS

```
--tlsMode requireTLS --tlsCertificateKeyFile ../opensslKeys/server.pem --
tlsCAFile
    ../opensslKeys/ca.pem
```

OR

Create a config file for each node of MongoDB cluster with relevant node information.

Sample configuration file:

```
net:
  bindIp: <hostname of node>
  port: <port number of node>
  tls:
    mode: requireTLS
    certificateKeyFile: /certificates/server.pem
    CAFile: /certificates/ca.pem
    clusterFile: /certificates/server.pem

replication:
  replSetName: "rs0"

storage:
  dbPath: /data/d1

security:
  authorization: enabled
  clusterAuthMode: x509
```

Stop the running node of MongoDB and start each node of multi-node cluster of MongoDB individually with the following config file.

```
mongod -config node1.conf
mongod -config node2.conf
mongod -config node3.conf
```

Initialize replica Set for MongoDB cluster

Connect MongoDB cluster with TLS/SSL enabled using MongoDB shell and initialize the replica set with the following command:

```
mongosh "mongodb://host-name:port/?authMechanism=MONGODB-X509&tls=true" --
tlsCertificateKeyFile=certs/client.pem --tlsCAFile=certs/ca.pem
```

Alternatively, use this connection string:

```
mongosh "mongodb://hostname:port/?
authMechanism=MONGODBX509&tls=true&tlsCertificateKeyFile=/path/to/
certificates/client.pem&tlsCAFile=/path/to/certificates/ca.pem"
```

Once connected with MongoDB shell, initialize the replica set with the following command:

```
rs.initiate({ _id: "rs1", members:
[
{ id: 0, host: "localhost:27017" },
{ id: 1, host: "localhost:27018" },
{ id: 2, host: "localhost:27019" }
] });
```

10.2.30.6.4 MongoDB Extract and Replicat Configuration

For secure connection to MongoDB cluster with X509 authentication and authorization enabled, the connection URI must include parameters for the authentication type, client certificate, and CA certificate. Also, we need to pass the client and server trust-store/key-store files and their passwords in JVM boot options.

Connection URI format

```
mongodb://hostname:port/?
authMechanism=MONGODBX509&tls=true&tlsCertificateKeyFile=/path/to/
certificates/client.pem&tlsCAFile=/path/to/certificates/ca.pem
```

JVM boot options

```
Jvm.bootoptions= -Xms512m -Xmx4024m -Xss32m -Djavax.net.ssl.trustStore=../
mcopensslkeys /server.pkcs12 -Djavax.net.ssl.trustStorePassword=<trust-store
password> -Djavax.net.ssl.keyStore=../mcopensslkeys/client.pkcs12 -
Djavax.net.ssl.keyStorePassword=<key-store password>
```

Creation of Credentials Store

```
add credentialstore user mongodb://hostname:port/?
authMechanism=MONGODBX509&tls=true&tlsCertificateKeyFile=/path/to/
certificates/client.pem&tlsCAFile=/path/to/certificates/ca.pem password root
alias mongo
```

OR

```
alter credentialstore add user mongodb://hostname:port/?
authMechanism=MONGODBX509&tls=true&tlsCertificateKeyFile=/path/to/
certificates/client.pem&tlsCAFile=/path/to/certificates/ca.pem password root
alias mongo
```

Note

The length of connectionString should not exceed 256.

Extract configuration

```
EXTRACT emongo
EXTTRAIL mn
SOURCEDB USERIDALIAS mongo DOMAIN OracleGoldenGate
JVMOPTIONS BOOTOPTIONS -Xms512m -Xmx4024m -Xss32m -Djavax.net.ssl.trustStore=/
certs/server.pkcs12 -Djavax.net.ssl.trustStorePassword=pass -
Djavax.net.ssl.keyStore=/certs/client.pkcs12 -
Djavax.net.ssl.keyStorePassword=pass CLASSPATH $OGG_HOME/ggjava/ggjava.jar:/
DependencyDownloader/dependencies/mongodb_capture_5.0.1/*
no_abend_on_ddl
TABLE srcDB.*;
```

Replicat configuration

```
gg.handlerlist=mongodb
gg.handler.mongodb.type=mongodb
gg.handler.mongodb.clientURI=mongodb://hostname:port/?
authMechanism=MONGODBX509&tls=true&tlsCertificateKeyFile=/path/to/
certificates/client.pem&tlsCAFile=/path/to/certificates/ca.pem
gg.classpath=/DependencyDownloader/dependencies/mongodb_5.0.1/*
jvm.bootoptions=-Xms512m -Xmx4024m -Xss32m -Djavax.net.ssl.trustStore=/tmp/
certs/server.pkcs12 -Djavax.net.ssl.trustStorePassword=pass -
Djavax.net.ssl.keyStore=/tmp/certs/client.pkcs12 -
Djavax.net.ssl.keyStorePassword=pass
```

10.2.30.7 Reviewing Sample Configurations

Basic Configuration

The following is a sample configuration for the MongoDB Handler from the Java adapter properties file:

```
gg.handlerlist=mongodb
gg.handler.mongodb.type=mongodb
```

```
#The following handler properties are optional.
#Refer to the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA)
documentation
#for details about the configuration.
#gg.handler.mongodb.clientURI=mongodb://localhost:27017/
```

```
#gg.handler.mongodb.WriteConcern={w:value, wtimeout: number }
#gg.handler.mongodb.BulkWrite=false
#gg.handler.mongodb.CheckMaxRowSizeLimit=true

goldengate.userexit.timestamp=utc
goldengate.userexit.writers=javawriter
javawriter.stats.display=TRUE
javawriter.stats.full=TRUE
gg.log=log4j
gg.log.level=INFO
gg.report.time=30sec

#Path to MongoDB Java driver.
# maven co-ordinates
# <dependency>
# <groupId>org.mongodb</groupId>
# <artifactId>mongo-java-driver</artifactId>
# <version>3.10.1</version>
# </dependency>
gg.classpath=/path/to/mongodb/java/driver/mongo-java-driver-3.10.1.jar
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/ggjava.jar:./dirprm
```

Oracle or MongoDB Database Source to MongoDB, AJD, and ATP Target

You can map an Oracle or MongoDB Database source table name in uppercase to a table in MongoDB that is in lowercase. This applies to both table names and schemas. There are two methods that you can use:

Create a Data Pump

You can create a data pump before the Replicat, which translates names to lowercase. Then you configure a MongoDB Replicat to use the output from the pump:

```
extract pmp
exttrail ./dirdat/le
map RAMOWER.EKKN, target "ram"."ekkn";
```

Convert When Replicating

You can convert table column names to lowercase when replicating to the MongoDB table by adding this parameter to your MongoDB properties file:

```
gg.schema.normalize=lowercase
```

10.2.30.8 MongoDB to AJD/ATP Migration

- [Overview](#)
- [Configuring MongoDB handler to Write to AJD/ATP](#)
- [Steps for Migration](#)
- [Best Practices](#)

10.2.30.8.1 Overview

Oracle Autonomous JSON Database (AJD) and Autonomous Database for transaction processing also uses wire protocol to connect. Wire protocol has the same MongoDB CRUD APIs.

10.2.30.8.2 Configuring MongoDB handler to Write to AJD/ATP

Basic configuration remains the same including optional properties mentioned in this chapter.

The handler uses same protocol (mongodb wire protocol) and same driver jar for Autonomous databases as that of mongodb for performing all operation in target agnostic manner for performing the replication. The properties can also be used for any of the supported targets.

The following is a sample configuration for the MongoDB Handler for AJD/ATP from the Java adapter properties file:

```
gg.handlerlist=mongodb
gg.handler.mongodb.type=mongodb
#URL mentioned below should be an AJD instance URL
gg.handler.mongodb.clientURI=mongodb://[username]:[password]@[url]?
authSource=$external&authMechanism=PLAIN&ssl=true
#Path to MongoDB Java driver. Maven co-ordinates
# <dependency>
# <groupId>org.mongodb</groupId>
# <artifactId>mongo-java-driver</artifactId>
# <version>3.10.1</version>
# </dependency>
gg.classpath=/path/to/mongodb/java/driver/mongo-java-driver-3.10.1.jar
javawriter.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/ggjava.jar:./dirprm
```

10.2.30.8.3 Steps for Migration

To migrate from MongoDB to AJD, first it is required to run initial load. Initial load comprises inserts operations only. After running initial load, start CDC which keeps the source and target database synchronized.

1. Start CDC extract and generate trails. Do not start replicat to consume these trail files.
2. Start Initial load extract and wait for initial load to complete.
3. Create a new replicat to consume the initial load trails generated in Step 2. Wait for completion and then stop replicat.
4. Create a new replicat to consume the CDC trails. Configure this replicat to use HANDLECOLLISIONS and then start replicat.
5. Wait for the CDC replicat (Step 4) to consume all the trails, check replicat lag, and replicat RBA to ensure that the CDC replicat has caught up. At this point, the source and target databases should be in sync.
6. Stop the CDC replicat, remove HANDLECOLLISIONS parameter, and then restart the CDC replicat.

10.2.30.8.4 Best Practices

For migration from mongoDB to Oracle Autonomous Database (AJD/ATP), following are the best practices:

1. Before running CDC, ensure to run initial load, which loads the initial data using insert operations.
2. Use bulk mode for running mongoDB handler in order to achieve better throughput.
3. Enable handle-collision while migration to allow replicat to handle any collision error automatically.

4. In order to insert missing update, ensure to add the `INSERTMISSINGUPDATES` property in the `.prm` file.

10.2.30.9 Configuring an Initial Synchronization of Extract for a MongoDB Source Database using Precise Instantiation

This article how to perform an initial synchronization of data from a MongoDB source database to a MongoDB/ORDS target using the **Precise Instantiation** method. This approach eliminates need for initial load enhancing performance and ensures that no collisions occur in the target Replicat, optimizing performance by eliminating the need for collision handling, which can degrade performance.

This **precise instantiation** approach involves taking a snapshot of the MongoDB source database using the MongoDB dump utility, and synchronizing the first operation of this initial dump with the starting position of Change Data Capture (CDC) on the Extract side. This process ensures no data loss or duplication between the dumped data and the subsequent CDC trail generated by the Extract.

- [Overview of the Synchronization Process](#)
- [Prerequisites](#)
- [Step-by-Step Instructions](#)

10.2.30.9.1 Overview of the Synchronization Process

1. **MongoDB Dump Utility:** The **mongodump** utility is used to create a snapshot of the source MongoDB database. It can dump documents, metadata, and index definitions to a binary archive. When the `--oplog` option is enabled, it also writes an entry in the `oplog.bson` file, capturing all operations (including any that occur during the dump) with their timestamps.
2. **Analyzing the Oplog:** The `oplog.bson` file contains a series of operations and their timestamps. This file is analyzed to identify the first and last operation during the dump. If no operations occurred during the dump process, the `oplog.bson` contains only a `noop` entry. The timestamps for the first and last operations (or the first `noop` entry if no operations occurred) are critical for synchronizing the CDC Extract.
3. **MongoDB Restore Utility:** After the dump is completed, the **mongorestore** utility is used to restore the data, metadata, and index definitions to the target MongoDB instance. The `--nsInclude` option allows for selective restoration of specific collections on target.
4. **CDC Extract and Replicat:** The CDC Extract is started from the first operation timestamp noted in the `oplog.bson` file. The Replicat is configured to apply changes from the trail file, starting from the last operation timestamp from the `oplog.bson`, configured in the `oplogReplayLastLsn` property ensuring no need for `handlecollision`.

These steps ensure that no operation is missed and avoid any duplication on target.

10.2.30.9.2 Prerequisites

Ensure the following:

1. **MongoDB source version** is 5 or higher.
2. **Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 23.7** or above is installed and configured for both the source MongoDB and target database instances.

3. The **MongoDB Database Tools** (including **mongodump** and **mongorestore**) are installed, and the respective directory paths are added to the path environment variable.

10.2.30.9.3 Step-by-Step Instructions

1. Run **MongoDB Dump Utility**. Run the **mongodump** command with the **--oplog** option to create a snapshot of the source MongoDB database:

```
$ ./bin/mongodump --uri="mongodb://localhost:27021" --oplog -v
```

Output:

```
2025-02-28T07:24:53.260+0000    getting most recent oplog timestamp
2025-02-28T07:24:53.268+0000    writing admin.system.version to dump/admin/
system.version.bson
2025-02-28T07:24:53.269+0000    done dumping admin.system.version (1 document)
2025-02-28T07:24:53.270+0000    dumping up to 4 collections in parallel
2025-02-28T07:24:53.271+0000    writing testDB.coll1 to dump/testDB/coll1.bson
2025-02-28T07:24:53.272+0000    writing testDB.coll2 to dump/testDB/coll2.bson
2025-02-28T07:24:53.272+0000    writing testDB.coll3 to dump/testDB/coll3.bson
2025-02-28T07:24:53.303+0000    done dumping testDB.coll3 (10000 documents)
2025-02-28T07:24:53.313+0000    done dumping testDB.coll1 (10000 documents)
2025-02-28T07:24:53.326+0000    done dumping testDB.coll2 (10000 documents)
2025-02-28T07:24:53.328+0000    writing captured oplog to
2025-02-28T07:24:53.328+0000    dumped 7 oplog entries
```

This generates a dump folder containing the binary archive data file in the following: dump/database-name/collection-name/collection-name.bson for all databases and collections and an oplog.bson directly under dump folder. To inspect the contents of the oplog.bson file (which is in binary format and not human readable), you can convert it to JSON using the **bsondump** utility:

```
$ ./bin/bsondump --pretty --outFile /path/to/oplog.json dump/oplog.bson
```

Output:

```
2025-02-28T07:25:03.346+0000    7 objects found
```

2. Extract Timestamps Using **OplogLSN.sh**.

Run the **OplogLSN.sh** script to extract the first and last operation timestamps from the oplog.bson file by providing the path to oplog.bson file as an argument as follows: **\$. /oplogLSN.sh /path/to/dump/oplog.bson**

The output provides the timestamps of the **first** and **last** operation (or the first noop if no operation occurred during the dump process)

Output:

```
2025-02-27T13:53:00.885+0000 1 objects found
First LSN: 1740663867.1
Last LSN: 1740663946.211
```

3. Inspect Oplog Entries.

If there were any incoming operations during the dump, then the oplog.bson file contains entries for those operations, each with a timestamp. You can use **OplogLSN.sh** to capture the first and last operation timestamps OR convert it to JSON file for manual inspection, as shown in the previous step.

4. Run MongoDB Restore Utility.

After you have the necessary timestamps, use the **mongorestore** utility to restore the selected collections from the dump to the target MongoDB instance:

```
$ ./mongorestore --uri="mongodb://localhost:27021" --nsInclude=testDB.coll1 --
nsInclude=testDB.coll2 /path/to/dump -v
```

```

Output:
2025-02-28T07:26:08.760+0000    using write concern: &{majority <nil> 0s}
2025-02-28T07:26:08.785+0000    using default 'dump' directory
2025-02-28T07:26:08.785+0000    preparing collections to restore from
2025-02-28T07:26:08.785+0000    found collection admin.system.version bson to
restore to admin.system.version
2025-02-28T07:26:08.785+0000    found collection metadata from admin.system.version
to restore to admin.system.version
2025-02-28T07:26:08.785+0000    don't know what to do with file "dump/oplog.json",
skipping...
2025-02-28T07:26:08.785+0000    found collection testDB.coll1 bson to restore to
testDB.coll1
2025-02-28T07:26:08.785+0000    found collection metadata from testDB.coll1 to
restore to testDB.coll1
2025-02-28T07:26:08.785+0000    found collection testDB.coll2 bson to restore to
testDB.coll2
2025-02-28T07:26:08.785+0000    found collection metadata from testDB.coll2 to
restore to testDB.coll2
2025-02-28T07:26:08.785+0000    reading metadata for testDB.coll1 from dump/testDB/
coll1.metadata.json
2025-02-28T07:26:08.785+0000    reading metadata for testDB.coll2 from dump/testDB/
coll2.metadata.json
2025-02-28T07:26:08.786+0000    creating collection testDB.coll1 with no metadata
2025-02-28T07:26:08.791+0000    creating collection testDB.coll2 with no metadata
2025-02-28T07:26:08.827+0000    restoring testDB.coll1 from dump/testDB/coll1.bson
2025-02-28T07:26:08.843+0000    restoring testDB.coll2 from dump/testDB/coll2.bson
2025-02-28T07:26:09.144+0000    finished restoring testDB.coll1 (10000 documents, 0
failures)
2025-02-28T07:26:09.204+0000    finished restoring testDB.coll2 (10000 documents, 0
failures)
2025-02-28T07:26:09.204+0000    no indexes to restore for collection testDB.coll1
2025-02-28T07:26:09.204+0000    no indexes to restore for collection testDB.coll2
2025-02-28T07:26:09.204+0000    20000 document(s) restored successfully. 0
document(s) failed to restore.

```

This command restores the data, metadata, and index definitions for the specified collection (for Example: coll1 and coll2 in database testDB as shown above) to the target MongoDB/ORDS instance.

5. Start MongoDB CDC Extract.
Start the MongoDB CDC Extract process from the first operation timestamp (LSN position) extracted from the `oplog.bson` file. This will ensure the CDC Extract begins capturing operations that occur after the dump process started.

Figure 10-2 Start MongoDB CDC Extract

```
# Start CDC Extract from the first operation timestamp
```

Begin Position in Trail ▼

Log Sequence Number (LSN):
1740663867.1

Enter 1 to 100 characters.

6. Configure MongoDB Replicat.

Configure the MongoDB Replicat with the generated CDC trail file and set the `oplogReplayLastLsn` property to the last operation timestamp from the `oplog.bson` file. This ensures the Replicat runs in `oplog-replay` mode, avoiding any collisions. Once the last timestamp is processed, it continues in normal mode.

```
# Set the oplogReplayLastLsn property in Replicat configuration. Either provide the  
last timestamp or just the path to oplog.bson file (and it figures out the last  
timestamp automatically).
```

```
gg.handler.mongodb.oplogReplayLastLsn=1740663946.211
```

This configuration guarantees precise initiation, with no data loss or duplication, and eliminates the need for costly collision handling in the Replicat.

Note

Clean-up before mongo-dump: The existing dump folder created inconsistent state of data at the time of mongo restore. Ensure that the existing dump folder is cleaned up before taking a new dump. It is very important to cleanup existing dump folder by deleting it and its contents before taking a new dump.

MongoDB restore: Multiple collections of a database can be replicated using multiple `--nsInclude` option in `mongorestore` command. However, the ORDS multiple databases cannot be restored using multiple `--nsIncludeMultiple` restore commands are required, one for each database need to be restored.

Recommended version of mongodb-database-tools is 100.10.0 or below. Latest version is not stable yet.

10.2.30.10 Delivery to Oracle AI Database JSON Collection Tables

This topic outlines the steps for installing Oracle AI Database 26ai Free, configuring the environment, creating a Pluggable Database (PDB) for Oracle REST Data Services (ORDS) integration, installing and setting up ORDS and use that to connect to the Oracle Database using Oracle Database MongoDB API using Oracle GoldenGate for Distributed Applications and Analytics MongoDB Handler.

Note

You can also capture from MongoDB source and deliver to Oracle JSON Collection Table (JCT) and Oracle JSON Duality View (JDV),

This topic contains the following:

- [Install Oracle Database 23c Free](#)
- [Connect to Oracle Database](#)
- [Create a New Pluggable Database \(PDB\)](#)
- [Create a User for ORDS](#)
- [Generate SSL Certificates](#)
- [Oracle REST Data Services \(ORDS\) Installation and Setup](#)

- [MongoDB Handler Configuration and Setup](#)
- [Error Handling](#)

10.2.30.10.1 Install Oracle Database 23c Free

Download Oracle AI Database 26ai Free:

1. Go to the official Oracle download page: [Oracle AI Database 26ai Free \(https://www.oracle.com/in/database/free/get-started/\)](https://www.oracle.com/in/database/free/get-started/).
2. Download the RPM package `oracle-ai-database-free-26ai-23.26.0-1.el8.x86_64.rpm` for Oracle Linux 8 (OL8) machines.
3. Install the RPM Package:
 - a. Run the following command to install Oracle Database Free `sudo dnf install -y oracle-database-free*`
Expected Output

```
Package: oracle-ai-database-free-26ai
Version: 1.0-1
Architecture: x86_64
Repository: @commandline
Size: 1.3 GB
```
4. Configure Oracle Database: After installation, configure the database with: `sudo /etc/configure. init.d/oracle-free-26ai`. You will be prompted to enter passwords for SYS, SYSTEM, and PDBADMIN users. Follow the on-screen instructions to complete configuration.

10.2.30.10.2 Connect to Oracle Database

Download Oracle Database Free 26ai:

1. Set Environment Variable: `export ORACLE_HOME=/opt/oracle/product/26ai/dbhomeFree/`.
2. Verify Installation: Connect to the Oracle Database to ensure installation is successful: `/opt/oracle/product/26ai/dbhomeFree/bin/sqlplus sys/Admin01@localhost:1521/FREE as sysdba`.

10.2.30.10.3 Create a New Pluggable Database (PDB)

1. Log in to SQL*Plus as SYSDBA: `/opt/oracle/product/23ai/dbhomeFree/bin/sqlplus sys/Admin01@localhost:1521/FREE as sysdba`.
2. Check Current Container and PDBs:

```
show con_name;
show pdbs;
```

Expected Output:

```
CON_NAME
-----
CDB$ROOT

CON_ID  CON_NAME      OPEN MODE  RESTRICTED
-----  -
2       PDB$SEED      READ ONLY  NO
3       FREEPDB1      READ WRITE NO
```

3. Create a New PDB for ORDS:

```
CREATE PLUGGABLE DATABASE pdbords ADMIN USER pdbordsadmin IDENTIFIED BY
Admin01 FILE_NAME_CONVERT = ('/opt/oracle/oradata/FREE/pdbseed/', '/scratch/
oracle/pdbseed_ords/');
```

Note

Precreate directory /scratch/oracle/pdbseed_ords/

4. Open the New PDB and verify: `ALTER PLUGGABLE DATABASE pdbords OPEN;SHOW pdbs;`
Expected Output:

CON_ID	CON_NAME	OPEN MODE	RESTRICTED
2	PDB\$SEED	READ ONLY	NO
3	FREEPDB1	READ WRITE	NO
5	PDBORDS	READ WRITE	NO

5. Set Container to PDBORDS:

```
ALTER SESSION SET CONTAINER = pdbords;
SHOW con_name;
```

10.2.30.10.4 Create a User for ORDS

1. Create ORDS User:

```
CREATE USER ordsuser IDENTIFIED BY Admin01;
GRANT CONNECT, RESOURCE, DBA TO ordsuser;
```

This user will be used for MongoDB integration through ORDS.

2. Connect as the New User:

```
/opt/oracle/product/26ai/dbhomeFree/bin/sqlplus ordsuser/Admin01@localhost:1521/
PDBORDS
```

10.2.30.10.5 Generate SSL Certificates

1. Create a directory for certificates:

```
mkdir certs
cd certs/
```

2. Generate the CA's Private Key. The following command will generate a private key for your Certificate Authority (CA). Remember the password used here, as it will be needed to sign other certificates.

```
openssl genrsa -aes256 -out ca-key.pem 4096
```

3. Create the CA's Public Certificate
Generate a self-signed public certificate for your CA, which acts as the Certificate Authority.

```
openssl req -new -x509 -sha256 -days 365 -key ca-key.pem -out ca.pem
```

4. View the CA Certificate's Details. Verify the details of the CA certificate with the following command:

```
openssl x509 -in ca.pem -text
```

5. Generate the Client's Private Key
Create a private key for the client:

```
openssl genrsa -out client-key.pem 4096
```

6. Create a Certificate Signing Request (CSR) Generate a CSR for the client certificate. Make sure the Common Name (CN) matches the hostname where the certificate will be used.

```
openssl req -new -sha256 -subj "/CN=oraclelinux8" -key client-key.pem -out client.csr
```

7. Generate the Client Certificate Using the CA. Add the IP addresses and DNS names that match your current environment to extfile.cnf. Ensure they are consistent with the hostname details in your hosts file.

Hosts File Example:

```
127.0.0.1 localhost
100.01.01.01 phoenix.dev3sub2phx.oraclevcn.com
```

Create extfile.cnf:

```
echo
"subjectAltName=DNS:*.oraclevcn.com,DNS:phoenix.dev3sub2phx.oraclevcn.com,IP:100.01.01.01" >> extfile.cnf
```

8. Sign the Certificate:

```
openssl x509 -req -sha256 -days 365 -in client.csr -CA ca.pem -CAkey ca-key.pem -out client.pem -extfile extfile.cnf -CAcreateserial
```

9. Verify the Client Certificate.

Use the following command to verify the client certificate:

```
openssl verify -CAfile ca.pem -verbose client.pem
```

10. Combine Certificates for Full Chain.

Create a full chain certificate by combining the client certificate and the CA certificate:

```
cat client.pem > fullchain.pem
cat ca.pem >> fullchain.pem
```

11. Configure ORDS to Use Your SSL Certificate.

Some systems require certificates in DER format instead of PEM.

Convert the Client Key and Certificate to DER Format:

```
openssl pkcs8 -topk8 -inform PEM -outform DER -in client-key.pem -out client-key.der -nocrypt
openssl x509 -inform PEM -outform DER -in client.pem -out client.der
```

10.2.30.10.6 Oracle REST Data Services (ORDS) Installation and Setup

1. **Download ORDS.**

- a. Download the file ords-24.3.0.262.0924.zip from Oracle's website: [ORDS Download](#)

- b. Unzip the downloaded file.

```
mkdir certs
cd certs/
```

2. **Setup ORDS:**

- a. Set Environment Variables.

```
export ORDS_HOME=/scratch/ords/
export ORDS_CONFIG=/scratch/ords/ordsconfig
export PATH=${ORDS_HOME}/bin:${PATH}
export _JAVA_OPTIONS="-Xms1126M -Xmx1126M"
```

Note

Requires JAVA11 or above. Unset ORACLE_HOME if set any.

- b. Install ORDS. Run the following command:
ords install

- Choose the following options during installation:
 - Connection Type: Basic
 - Database: localhost:1521/PDBORDS
 - Protocol: HTTPS
 - Port: 8443
 - Certificate Type: Use Self-Signed Certificate with fully qualified hostname

Expected Output:

```
-bash-4.4$ ords install
```

```
Enter a number to update the value or select option A to Accept and Continue
```

```
[1] Connection Type: Basic
```

```
[2] Basic Connection: HOST=localhost PORT=1521 SERVICE_NAME=PDBORDS
```

```
Administrator User: SYS AS SYSDBA
```

```
[3] Database password for ORDS runtime user (ORDS_PUBLIC_USER): <generate>
```

```
[4] ORDS runtime user and schema tablespaces: Default: SYSAUX Temporary TEMP
```

```
[5] Additional Feature: Database Actions
```

```
[6] Configure and start ORDS in Standalone Mode: Yes
```

```
[7] Protocol: HTTPS
```

```
[8] HTTPS Port: 8443
```

```
[9] Certificate Type: Use Self-Signed Certificate
```

```
[10] SSL Hostname: phoenix.dev3sub2phx.oraclevcn.com
```

```
[A] Accept and Continue - Create configuration and Install ORDS in the  
database
```

```
[Q] Quit - Do not proceed. No changes
```

```
Choose [A]: A
```

- c. Enable ORDS Schema. Connect to the database and grant necessary permissions:

```
/opt/oracle/product/26ai/dbhomeFree/bin/sqlplus ordsuser/Admin01@localhost:1521/  
PDBORDS
```

```
SQL> grant soda_app, create session, create table, create view, create sequence,  
create procedure, create job, unlimited tablespace to ordsuser;
```

```
SQL> exec ords.enable_schema;
```

- d. Configure Certificates and MongoDB.


```
ords config set standalone.https.cert /scratch/ords/certs/client.der
ords config set standalone.https.cert.key /scratch/ords/certs/client-key.der
ords config set mongo.enabled true
ords config set mongo.port 27040
```

3. Start ORDS

- a. Run the following command to start ORDS: `ords serve`

Expected Output:

```
-bash-4.4$ ords serve
```

```
Picked up _JAVA_OPTIONS: -Xms1126M -Xmx1126M
```

```
2024-11-04T10:53:10.119Z WARNING      Your configuration folder /scratch/ords/
ordsconfig is located in ORDS product folder.  Oracle recommends to use a different
configuration folder.
```

Refer to Oracle REST Data Services Documentation on how to setup your configuration folder.

```
ORDS: Release 24.3 Production on Mon Nov 04 10:53:10 2024
```

```
Copyright (c) 2010, 2024, Oracle.
```

Configuration:

```
/scratch/ords/ordsconfig
```

```
2024-11-04T10:53:10.642Z INFO          HTTP and HTTP/2 cleartext listening on host:
0.0.0.0 port: 8080
```

```
2024-11-04T10:53:10.643Z INFO          HTTPS and HTTPS/2 listening on host: 0.0.0.0
port: 8443
```

```
2024-11-04T10:53:10.679Z INFO          Disabling document root because the specified
folder does not exist: /scratch/ords/ordsconfig/global/doc_root
```

```
2024-11-04T10:53:10.680Z INFO          Default forwarding from / to contextRoot
configured.
```

```
2024-11-04T10:53:10.764Z SEVERE        ORAMLVERSION null
```

```
2024-11-04T10:53:10.775Z INFO          Oracle API for MongoDB listening on port: 27040
```

```
2024-11-04T10:53:10.775Z INFO          The Oracle API for MongoDB connection string is:
```

```
mongodb://[{user}:{password}@]localhost:27040/{user}?
authMechanism=PLAIN&authSource=$external&ssl=true&retryWrites=false&loadBalanced=true
```

10.2.30.10.7 MongoDB Handler Configuration and Setup

1. Add generated CA Certificate to Java Keystore.

- a. Add generated CA Certificate to Java Keystore To enable secure communication with MongoDB, the CA certificate (ca.pem) must be added to the Java keystore. Run the following command:

```
keytool -import -trustcacerts -keystore /scratch/oui_21.15/
ogg_install_home/jdk/jre/lib/security/cacerts -storepass changeit -alias MiCA -file /
scratch/ca.pem -noprompt
```

The certificate should now be added to the keystore successfully.

Note

Use cacerts from OGGDAA installation directory.

2. **MongoDB Connection URI:** To connect to MongoDB securely, use the following connection URI. Make sure that paths for `tlsCAFile` and `tlsCertificateKeyFile` are correct and match your environment. Set Environment Variables.

```
mongodb://ordsuser:Admin01@phoenix.dev3sub2phx.oraclevcn.com:27040/ordsuser?
authMechanism=PLAIN&authSource=$external&ssl=true&retryWrites=false&loadBalanced=true
&tlsCAFile=/scratch/ca.pem
```

Syntax:

```
mongodb://[{user}:{password}@]localhost:27040/{user}?
authMechanism=PLAIN&authSource=$external&ssl=true&retryWrites=false&loadBalanced=true
```

3. **Create Table Space ORDS**

- a. **Connect to SQL Plus:**

```
/opt/oracle/product/26ai/dbhomeFree/bin/sqlplus ordsuser/Admin01@localhost:1521/
PDBORDS
```

- b. **Create Tablespace**

```
CREATE TABLESPACE jcttest
DATAFILE '/scratch/ords/tablespace/jct_datafile.dbf'
SIZE 100M
AUTOEXTEND ON
NEXT 10M
MAXSIZE UNLIMITED
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
```

Note

Precreate /scratch/ords/tablespace directory.

- c. **Set Default Tablespace:**

```
ALTER USER ordsuser DEFAULT TABLESPACE JCTTEST;
```

4. **Configure Replication**

- a. To configure the MongoDB handler, create a prm file for the replicat as:

```
REPLICAT JCT
MAP *.* , TARGET ordsuser.*; // here target schema is the ords user that we
created.
```

- b. Create a properties file for the Replicat JCT.

```
gg.handlerlist=mongodb
gg.handler.mongodb.type=mongodb
#TODO: MongoDB Client URI (replace with the actual MongoDB URI)
gg.handler.mongodb.clientURI=mongodb://
ordsuser:Admin01@phoenix.dev3sub2phx.oraclevcn.com:27040/ordsuser?
authMechanism=PLAIN&authSource=$external&ssl=true&retryWrites=false&loadBalanced=
true&tlsCAFile=/scratch/ca.pem
#TODO: Path to MongoDB Java driver
```

```
gg.classpath=/scratch/oui_21.15/ogg_install_home/opt/DependencyDownloader/  
dependencies/mongodb_5.0.0/*
```

5. Verify Data in ORDS SQL Developer

After setting up replication, you can verify the data in SQL Developer:

Login URL: SQL Developer URL (<https://phoenix.dev3sub2phx.oraclevcn.com:8443/ords/sql-developer>)

Credentials: ordsuser/Admin01

```
SELECT DATA, RESID, ETAG FROM ORDSUSER.JSONCT;
```

The DATA column should hold the JCT records from MongoDB.

Note

- If the JCT table is not present, the MongoDB handler will create it by default.
- You may also create the target JCT table manually and map it in the handler configuration.

Syntax to Create JSON Collection Table:

```
CREATE JSON COLLECTION TABLE JSONCT;
```

10.2.30.10.8 Error Handling

1. If you come across the error: JSON type cannot be used in non-automatic segment space management tablespace 'SYSTEM', run the following commands to assign a default tablespace to ordsuser.

Connect to SQL Plus:

```
/opt/oracle/product/26ai/dbhomeFree/bin/sqlplus ordsuser/Admin01@localhost:1521/  
PDBORDS
```

Set Default Tablespace:

```
ALTER USER ordsuser DEFAULT TABLESPACE JCTTEST;
```

This sets JCTTEST as the default tablespace for the ordsuser schema.

10.2.30.11 MongoDB Handler Client Dependencies

What are the dependencies for the MongoDB Handler to connect to MongoDB databases?

Oracle GoldenGate requires version 4.6.0 MongoDB reactive streams for integration with MongoDB. You can download this driver from: <https://search.maven.org/artifact/org.mongodb/mongodb-driver-reactivestreams>

Note

If the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) version is 21.7.0.0.0 and below, the driver version is [MongoDB Java Driver 3.12.8](#). For Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) versions 21.8.0.0.0 and above, the driver version is [MongoDB Java Driver 4.6.0](#).

- [MongoDB Java Driver 4.6.0](#)
- [MongoDB Java Driver 3.12.8](#)

10.2.30.11.1 MongoDB Java Driver 4.6.0

The required dependent client libraries are:

- bson-4.6.0.jar
- bson-record-codec-4.6.0.jar
- mongodb-driver-core-4.6.0.jar
- mongodb-driver-legacy-4.6.0.jar
- mongodb-driver-legacy-4.6.0.jar
- mongodb-driver-sync-4.6.0.jar

The Maven coordinates of these third-party libraries that are needed to run MongoDB replicat are:

```
<dependency>

    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-legacy</artifactId>
    <version>4.6.0</version>
</dependency>

<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>4.6.0</version>
</dependency>
```

Example

Download the latest version from Maven central at: <https://central.sonatype.com/artifact/org.mongodb/mongodb-driver-reactivestreams/4.6.0>.

10.2.30.11.2 MongoDB Java Driver 3.12.8

You must include the path to the MongoDB Java driver in the gg.classpath property. To automatically download the Java driver from the Maven central repository, add the following lines in the pom.xml file, substituting your correct information:

```
<!-- https://mvnrepository.com/artifact/org.mongodb/mongo-java-driver -->
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongo-java-driver</artifactId>
    <version>3.12.8</version>
</dependency>
```

10.2.31 Oracle AI Data Platform

Overview

Oracle AI Data Platform is a new product offering that brings together data lake, analytics, and orchestration services into one unified platform. With centralized governance, built-in

collaboration, and seamless integration across internal and external data sources, it helps enterprises to accelerate their data-driven decision-making and enables them to perform data analytics at scale with greater speed and efficiency.

- [Detailed Functionality](#)
- [Configuration](#)
- [Troubleshooting and Diagnostics](#)
- [Restrictions on Database Object Names](#)

10.2.31.1 Detailed Functionality

Overview

Replication to Oracle AI Data Platform uses the stage and merge data flow. The change data from the Oracle GoldenGate trails is staged in micro-batches at a temporary staging location (external stage) in OCI Object Storage. The staged records are then merged into the Oracle AI Data Platform target tables using a merge SQL statement. By default, staged records are merged in every 3 minutes.

- [Staging Location](#)
- [Database User Privileges](#)
- [Prerequisites](#)

10.2.31.1.1 Staging Location

The change data records from the Oracle GoldenGate trail files are formatted into Avro OCF (Object Container Format) and are then uploaded to the staging location.

Change data can be staged on the following object store:

- OCI Object storage

10.2.31.1.2 Database User Privileges

For replicating into the Oracle AI Data Platform the database user requires the following privileges:

- SELECT, CREATE, INSERT, UPDATE, and DELETE permissions on the target tables.
- SELECT, CREATE, ALTER, and DROP permissions on the external tables.

10.2.31.1.3 Prerequisites

- You must have Oracle Cloud Infrastructure account set up for Oracle AI Data Platform.
- Simba JDBC driver for Apache Spark. You can download the Simba JDBC driver from the cluster detail page of the Oracle AI Data Platform console. For more information, see [Simba JDBC Driver for Apache Spark](#).

10.2.31.2 Configuration

The Oracle AI Data Platform replication properties configuration is stored in the Replicat properties file.

- [Automatic Configuration](#)
- [Authentication to Oracle AI Data Platform](#)
- [Classpath Configuration](#)
- [INSERTALLRECORDS Support](#)
- [Operation Aggregation](#)
- [End-to-End Configuration](#)
- [Table Mapping](#)

10.2.31.2.1 Automatic Configuration

Oracle AI Data Platform replication involves configuring multiple components, such as the File Writer Handler, the OCI Event Handler, and the target Oracle AI Data Platform Event Handler.

The Automatic Configuration functionality helps you to configure these components automatically to minimize the manual configuration.

The properties modified by automatic configuration is also logged in the handler log file.

The parameter `gg.stage` determines the staging location.

To enable auto-configuration to replicate to the Oracle AI Data Platform target, set the parameter `gg.target=aidp`.

If `gg.stage` is not set, the OCI object storage will be used as the staging location.

The JDBC Metadata provider is also automatically enabled to retrieve target table metadata from the Oracle AI Data Platform.

Target tables are automatically created if missing.

- [File Writer Handler Configuration](#)
- [OCI Event Handler Configuration](#)
- [Oracle AI Data Platform Event Handler Configuration](#)

10.2.31.2.1.1 File Writer Handler Configuration

The File Writer Handler name is pre-set to the value `aidp` and its properties are automatically set to the required values for Oracle AI Data Platform.

10.2.31.2.1.2 OCI Event Handler Configuration

The OCI Event Handler name is pre-set to the value `oci` and must be configured to match your OCI configuration.

The following is an example of editing the OCI Event property:

```
gg.eventhandler.oci.bucketMappingTemplate=container1
```

Related Topics

- [Oracle AI Data Platform Event Handler Configuration](#)

10.2.31.2.1.3 Oracle AI Data Platform Event Handler Configuration

The Oracle AI Data Platform Event Handler name is pre-set to the value aidp.

The following are the configuration properties available for the Oracle AI Data Platform Event handler, the required ones must be changed to match your Oracle AI Data Platform configuration.

Table 10-37 Oracle AI Data Platform Event Handler Configuration properties

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .aidp.connectionURL	Required	jdbc:spark:// <server- hostname>;Spark ServerType=AIDP ;httpPath=<http - path>;<setting 1>=<value1>;<se tting2>=<value2 >;<settingN>=<v alueN>]	None	Specifies the JDBC URL to connect to Oracle AI Data Platform.
gg.eventhandler .aidp.maxConnec tions	Optional	Integer Value	10	You can use this parameter to control the number of concurrent JDBC database connections to the target database.
gg.eventhandler .aidp.connection nRetries	Optional	Integer Value	3	Specifies the number of times connections to the target data warehouse will be retried.
gg.eventhandler .aidp.connection nRetryIntervals econds	Optional	Integer Value	30	Specifies the delay in seconds between connection retry attempts.
gg.eventhandler .aidp.createTab le	Optional	True or False	True	If the parameter value is set to true. The missing target tables are automatically created.

Table 10-37 (Cont.) Oracle AI Data Platform Event Handler Configuration properties

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .aidp.detectMis singBaseRow	Optional	True or False	False	Diagnostic parameter to find the UPDATE operations without base row. If set to true, Replicat will ABEND if there are Confidential – Oracle Internal UPDATE operations without base row. These rows are collected into another table that can be investigated.
gg.eventhandler .aidp.dropStagi ngTablesOnShutd own	Optional	True or False	False	If the parameter value is set to true, the temporary staging tables created by Oracle GoldenGate will be dropped on replicat graceful stop.
gg.handler.aidp .fileRollInterv al	Optional	The default unit of measure is milliseconds. You can stipulate ms, s, m, h, to signify milliseconds, seconds, minutes, or hours respectively. Examples of legal values include 10000, 10000ms, 10s, 10m, or 1.5h. Values of 0 or lower indicate that file roll-out operation is disabled.	3m	The parameter determines how often the data will be merged into AIDP. Caution: Increasing the default value from 3m to higher value will increase the amount of data stored in the Replicat internal memory. This can cause out of memory errors and stop the Replicat process.
gg.eventhandler .aidp.deltaUniv ersalFormat	Optional	iceberg or " " (empty).	" " (empty)	When set to Iceberg, Replicat automatically creates target tables in Delta Lake Universal Format configured for Iceberg. This allows you to read Delta tables with Iceberg reader clients.

10.2.31.2.2 Authentication to Oracle AI Data Platform

The Oracle AI Data Platform JDBC connection URLs use the following format for authentication:

```
jdbc:spark://<Host>;SparkServerType=AIDP;httpPath=<http-path>[;<setting1>=<value1>;<setting2>=<value2>;<settingN>=<valueN>]
```

- [Oracle AI Data Platform JDBC Driver Compute Settings](#)
- [Connection Details for Oracle AI Data Platform Compute Cluster](#)
- [Simba JDBC Driver for Apache Spark](#)
- [Oracle AI Data Platform Authentication Methods](#)

10.2.31.2.2.1 Oracle AI Data Platform JDBC Driver Compute Settings

The Oracle AI Data Platform JDBC driver requires the following compute resource configuration settings.

Table 10-38 Oracle AI Data Platform JDBC Driver compute resource configuration settings

Setting	Description
Host	Specifies the Oracle AI Data Platform compute resource's Server hostname value.
SparkServerType	Specifies the Oracle AI Data Platform compute resource's spark server type, that is, AIDP.
httpPath	Specifies the Oracle AI Data Platform compute resource's HTTP path value.

10.2.31.2.2.2 Connection Details for Oracle AI Data Platform Compute Cluster

To get the connection details for the Oracle AI Data Platform compute cluster:

1. Log in to your Oracle AI Data Platform instance.
2. In the sidebar, select the required workspace and then click **Compute**.
3. From the list of available clusters, click the *target cluster*'s name.
4. Click on the **Connection Details** tab that mentions the *JDBC URL* details and the *driver details* associated with that cluster.
5. In the **JDBC** tab, the JDBC URL is available for the use.

10.2.31.2.2.3 Simba JDBC Driver for Apache Spark

Use the following steps to get the Simba JDBC driver for the Oracle AI Data Platform compute cluster:

1. Log in to your Oracle AI Data Platform instance.
2. Select the required workspace.

3. Click **Compute**.
4. Select the target cluster's name from the list of available clusters.
5. Click the **Connection Details** tab that mentions the *JDBC URL details* and the *driver details* associated with that cluster.
6. From the **JDBC** tab, select **Download JDBC Driver** option to obtain the JDBC driver jar.

10.2.31.2.2.4 Oracle AI Data Platform Authentication Methods

The Oracle AI Data Platform JDBC driver supports the following authentication methods:

OCI API Key

Use the following steps to generate the OCI API key and the corresponding profile:

1. In your OCI tenancy where AI Data Platform instance is present, navigate to the **User** icon available at the top-right corner.
2. Click **User Settings, Tokens & Keys, Add API Key**.
3. Click **Generate API key pair**.
4. Download the Public and Private keys.
5. Click **Add** to add the Public and Private keys.
6. From the displayed window, copy the configuration file preview and add the contents in the `~/.oci/config` file.

Sample OCI Configuration File

```
[DEFAULT]
user=ocid1.user.oc1..mockValue
fingerprint=mockFingerPrintValue
tenancy=ocid1.compartment.oc1..mockValue
region=us-phoenix-1
key_file=<path to your private keyfile>
```

Note

- The Simba JDBC driver has default configuration of the OCI configurations in the `~/.oci/config` file with the profile name as `DEFAULT`.
- If you change any of the property, provide the similar details in the JDBC URL using the keys, `OCIConfigFile` and `OCIProfile` to indicate the new configuration file path and new name of the OCI profile. For Example:

```
jdbc:spark://sample.url.com/
default;SparkServerType=AIDP;httpPath=cliservice/
mockPath;OCIConfigFile=/tmp/config;OCIProfile=MY_PROFILE
```

10.2.31.2.3 Classpath Configuration

Oracle AI Data Platform Event Handler uses the Simba JDBC driver for Apache Spark.

Ensure the following for Simba JDBC driver classpath configuration:

- The classpath includes the path to the JDBC driver.
- Include the OCI object store event handler's dependencies in the classpath.
- Edit the `gg.classpath` configuration parameter to include the path to the object store event handler dependencies and the Simba JDBC driver for Apache Spark.

10.2.31.2.4 INSERTALLRECORDS Support

To enable bulk insert operations for Replicat, set the `INSERTALLRECORDS` parameter in the Replicat parameter file (`.prm`). This optimizes data loading into the target table.

You can adjust the bulk insert batch size and frequency using the following File writer properties:

- `gg.handler.aidp.maxFileSize` with 1GB default value for batch size.
- `gg.handler.aidp.fileRollInterval` with 3 minutes default value for bulk insert frequency

10.2.31.2.5 Operation Aggregation

Operation aggregation combines multiple operations on the same row into a single output operation based on a threshold.

- [In-Memory Operation Aggregation](#)
- [SQL-Based Operation Aggregation](#)

10.2.31.2.5.1 In-Memory Operation Aggregation

By default, operation records are aggregated in-memory. You can adjust the merge interval using the `gg.handler.aidp.fileRollInterval` property. The default value for `gg.handler.aidp.fileRollInterval` property is 3 minutes.

① Note

The in-memory operation aggregation requires additional JVM memory configuration.

10.2.31.2.5.2 SQL-Based Operation Aggregation

Currently, Oracle AI Data Platform does not support the operation aggregation using SQL.

10.2.31.2.6 End-to-End Configuration

The following is an end-to-end configuration example which uses auto-configuration.

Oracle AI Data Platform with OCI Object Storage

```
# Properties file for Replicat
# Configuration to load GoldenGate trail operation records into AI Data
# Platform using OCI object store staging location.
# Note: Recommended to only edit the configuration marked as TODO
gg.target=aidp
```

```
# OCI Event handler.
#TODO: Edit the OCI region
gg.eventhandler.oci.region=<oci-region>
#TODO: Edit the OCI compartment OCID
gg.eventhandler.oci.compartmentID=<oci-compartment-ocid>
#TODO: Edit the OCI bucket name
gg.eventhandler.oci.bucketMappingTemplate=<oci-bucket-name>
#TODO: Edit the OCI Config Profile name
gg.eventhandler.oci.profile=<oci-profile-name>
#TODO: Edit the OCI Config file path
gg.eventhandler.oci.configFilePath=./oci/config

# AI Data Platform Event Handler.
#TODO: Edit JDBC ConnectionUrl
gg.eventhandler.aidp.connectionURL=<connection-url>

#TODO: Edit the classpath to include OCI Event handler dependencies and Simba
JDBC driver.
gg.classpath=/path/to/oci-dep/*:/path/to/simba-jdbc-driver/*
```

10.2.31.2.7 Table Mapping

If the MAP statement does not specify a target catalog, then the Oracle AI Data Platform workspace use hive as the default catalog. The event handler will use this as the default catalog during initialization and display the log message as, for example Connection catalog is set to [hive].

Table 10-39 Mapping Table

MAP statement in the Replicat parameter file	AIDP Catalog	AIDP Schema	AIDP Table
MAP SCHEMA_1.TABLE_1, TARGET "schema_1"."table_1 ";	Default catalog, that is, hive	schema_1	table_1
MAP DB_1.SCHEMA_1.TABLE _1, TARGET "db_1"."schema_1"." table_1"	db_1	schema_1	table_1

10.2.31.3 Troubleshooting and Diagnostics

- **Unsupported Oracle AI Data Platform datatypes:**
 1. ARRAY
 2. MAP
 3. STRUCT
- **Avoid Creating Bucket Mapping Templates with Dynamic Values:**

Avoid creating bucket mapping templates for OCI event handler that use dynamic values, as this may lead to the generation of multiple buckets for a single table. This can result in inconsistent results as the application is using the REFRESH SQL statement to update the external table with the new available staged file, so the location of all the staging files of a particular table should be consistent.

1. Examples of invalid bucket mapping templates:

- Using `test_run1_${tableName}_${currentTimestamp}` where `currentTimestamp` is replaced with a dynamic timestamp value.
For example, `2025-09-02_05-54-26.166`, resulting in multiple buckets, such as `test_run1_HISTORY_2025-09-02_05-54-26.166`,
`test_run1_HISTORY_2025-09-02_06-21-46.006`,
`test_run1_NEW_ORDER_2025-09-02_05-54-28.005`.

2. Examples of valid bucket mapping templates:

- Using a constant bucket mapping template, such as `testrun1`, which would ensure that there is a unique single path for each table within the same single bucket for the entire execution of the replicat.
- Using a template, such as `test_run1_${tableName}`, which would result in unique single bucket for each source table, such as `test_run1_HISTORY`,
`test_run1_NEW_ORDER`.

- **Co-existence of the components:**

When using an external stage location, the location/region of the machine where Replicat process is running and the object store's region will have an impact on the overall throughput of the apply process. For the best possible throughput, the components need to be located ideally in the same region or as close as possible.

- **DDL not applied on the target table :**

Oracle GoldenGate for Distributed Applications and Analytics does not support DDL replication

- **Handling Timestamp Values Before 1900-01-01:**

Oracle AI Data Platform currently do not support timestamp datatype values before 1900-01-01 by default due to its internal date representation.

To enable writing timestamp values before 1900-01-01, set the following property in your cluster configuration:

```
spark.sql.parquet.int96RebaseModeInWrite=CORRECTED
```

Restart the cluster to apply the change.

- Oracle recommends to use the Simba JDBC jar lean version included in the ZIP package downloaded from the Oracle AI Data Platform console. The Simba JDBC jar lean version includes shaded dependencies. These shaded dependencies help prevent dependency resolution conflicts between Oracle GoldenGate and the Simba JDBC driver, thereby ensuring smoother integration and compatibility. The lean Simba JDBC driver jar is available in the downloaded ZIP archive's `libs` directory.

10.2.31.4 Restrictions on Database Object Names

- **Catalog, Schema, and Table Names:** AIDP supports only names that comply with the Avro specification for identifier names: must start with ``[A-Za-z_]``, followed by ``[A-Za-z0-9_]``

- Additionally, all object names are also limited by the supported list of characters for the respective target.

10.2.32 OCI Streaming

Oracle Cloud Infrastructure Streaming (OCI Streaming) supports putting messages to and receiving messages using the Kafka client. Therefore, Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) can be used to publish change data capture operation messages to OCI Streaming.

You can use either the [Kafka Handler](#) or the [Kafka Connect Handler](#). The Kafka Connect Handler only supports using the JSON Kafka Connect converter. The Kafka Connect Avro converter is not supported because the Avro converter requires connectivity to a schema registry.

Note

The Oracle Streaming Service currently does not have a schema registry to which the Kafka Connect Avro converter can connect. Streams to which the Kafka Handlers or the Kafka Connect Handlers publish messages must be pre-created in Oracle Cloud Infrastructure (OCI). Using the Kafka Handler to publish messages to a stream in OSS which does not already exist results in a runtime exception.

- To create a stream in OCI, in the OCI console, select **Analytics**, click **Streaming**, and then click **Create Stream**. Streams are created by default in the **DefaultPool**.

Figure 10-3 Example Image of Stream Creation

The screenshot displays the Oracle Cloud console interface for creating a new stream. The 'Create Stream' modal is open, showing the following details:

- CHOOSE NEW STREAM POOL IN YDAMA:** A dropdown menu with 'DefaultPool' selected.
- STREAM NAME:** A text input field containing 'OGGBD-191002'.
- Stream Settings:**
 - RETENTION (IN HOURS):** A text input field with '24'. A note below states: 'Retention in hours must be between 1 - 672. Default value is 24 hours.'
 - NUMBER OF PARTITIONS:** A text input field with '1'. A note below states: 'The maximum number of partitions is based on the limits of your tenancy.'
 - TOTAL WRITE RATE (IN MB PER SECOND):** A text input field with '1 MB/s'. The label is marked as 'READ-ONLY'.
 - TOTAL READ RATE (IN MB PER SECOND):** A text input field with '2 MB/s'. The label is marked as 'READ-ONLY'.

The background shows the 'Analytics' sidebar with 'Streams' selected, and the 'US West (Phoenix)' region is indicated in the top right.

- The Kafka Producer client requires certain Kafka producer configuration properties to connect to OSS streams. To obtain this connectivity information, click the pool name in the OSS panel. If DefaultPool is used, then click **DefaultPool** in the OSS panel.

Figure 10-4 Example OSS Panel showing DefaultPool

OGGBD-191002

Produce Test Message Move Resource Add Tags Delete

Stream Information

Stream Name: OGGBD-191002

OCID: ...t5addq [Show](#) [Copy](#)

Compartment: rootpath/test_quality/compartmentName

Messages Endpoint: https://streaming.cloud.com

Stream Pool: [DefaultPool](#) [Move](#)

Settings

Number of partitions: 1

Retention: 24 hours

Read Throughput: 2 MB/s

Write Throughput: 1 MB/s

Recent Messages

Click Load Messages to consume 50 messages published in last minute

Load Messages

Key	Value	Offset	Partition

Figure 10-5 Example DefaultPool Properties

Home > Streaming > Stream Pools > Stream Pool Details

DefaultPool

ACTIVE

Resources

Streams

Create

Kafka Connection Settings for Stream Pool

BOOTSTRAP SERVERS READ-ONLY ⓘ

streaming.cloud.com:9092 [Copy](#)

SASL CONNECTION STRINGS READ-ONLY ⓘ

org.apache.kafka.security.LoginModule required username =
*rootpath/cloudservice/compartmentName@email.com/ocid.ph [Copy](#)

SECURITY PROTOCOL READ-ONLY ⓘ

SASL_SSL [Copy](#)

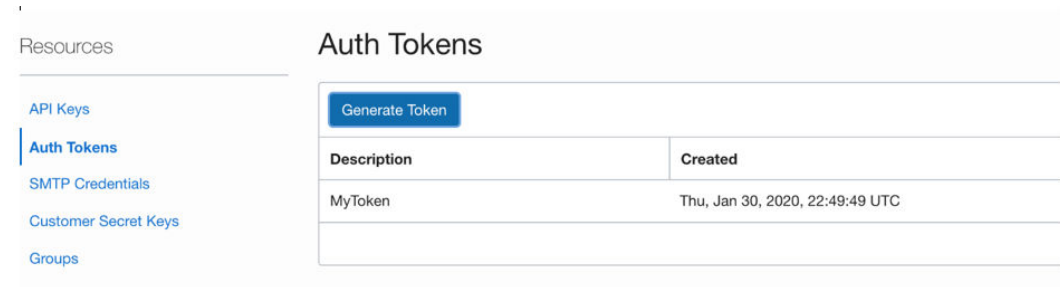
SECURITY MECHANISM READ-ONLY ⓘ

PLAIN [Copy](#)

Close

Name	Status	Created	Number of partitions	Read Throughput	Write Throughput
OGGBD-191002	Active	Tue, 25 Feb 2020 20:05:34 GMT	1	2 MB/s	1 MB/s

- The Kafka Producer also requires an AUTH-TOKEN (password) to connect to OSS. To obtain an AUTH-TOKEN go to the **User Details** page and generate an AUTH-TOKEN. AUTH-TOKENs are only viewable at creation and are not subsequently viewable. Ensure that you store the AUTH-TOKEN in a safe place.

Figure 10-6 Auth-Tokens

Once you have these configurations, you can publish messages to OSS.

For example, kafka.prm file:

```
replicat kafka
TARGETDB LIBFILE libggjava.so SET property=dirprm/kafka.properties
map *.* , target qatarget.*;
```

Example: kafka.properties file:

```
gg.log=log4j
gg.log.level=debug
gg.report.time=30sec
gg.handlerlist=kafkahandler
gg.handler.kafkahandler.type=kafka
gg.handler.kafkahandler.mode=op
gg.handler.kafkahandler.format=json
gg.handler.kafkahandler.kafkaProducerConfigFile=oci_kafka.properties
# The following dictates how we'll map the workload to the target OSS streams
gg.handler.kafkahandler.topicMappingTemplate=OGGBD-191002
gg.handler.kafkahandler.keyMappingTemplate=${tableName}
gg.classpath=/home/opc/dependencyDownloader/dependencies/kafka_2.2.0/*
jvm.bootoptions=-Xmx512m -Xms32m -Djava.class.path=ggjava/ggjava.jar:dirprm
```

Example Kafka Producer Properties (oci_kafka.properties)

```
bootstrap.servers=cell-1.streaming.us-phoenix-1.oci.oraclecloud.com:9092
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
value.serializer=org.apache.kafka.common.serialization.ByteArraySerializer
key.serializer=org.apache.kafka.common.serialization.ByteArraySerializer
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="paasdevgg/oracleidentitycloudservice/user.name@oracle.com/
ocid1.streampool.oc1.phx.amaaaaaa3p5c3vqa4hfyl7uv465pay4audmoajughhxlsgj7afc2an5u3xaq"
password="YOUR-AUTH-TOKEN";
```

To view the messages, click **Load Messages** in OSS.

Figure 10-7 Viewing the Messages

OGGBD-191002

Produce Test Message
Move Resource
Add Tags
Delete

Stream Information
Tags

Stream Information

Stream Name: OGGBD-191002
OCID: ...t5addq [Show](#) [Copy](#)
Compartment: rootpath/test_quality/compartmentName
Messages Endpoint: <https://streaming.cloud.com>
Stream Pool: [DefaultPool](#) [Move](#)

Settings

Number of partitions: 1
Retention: 24 hours
Read Throughput: 2 MB/s
Write Throughput: 1 MB/s

Recent Messages

Click Load Messages to consume 50 messages published in last minute

Load Messages

Key	Value	Offset	Partition	Created
Refresh to retrieve Recent Messages				

10.2.33 Oracle NoSQL

The Oracle NoSQL Handler can replicate transactional data from Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) to a target Oracle NoSQL Database.

This chapter describes how to use the Oracle NoSQL Handler.

- [Overview](#)
- [On-Premise Connectivity](#)
- [OCI Cloud Connectivity](#)
- [Oracle NoSQL Types](#)
- [Oracle NoSQL Handler Configuration](#)
- [Performance Considerations](#)
- [Operation Processing Support](#)
- [Column Processing](#)
- [Table Check and Reconciliation Process](#)
- [Oracle NoSQL SDK Dependencies](#)

10.2.33.1 Overview

Oracle NoSQL Database is a NoSQL-type distributed key-value database. It provides a powerful and flexible transaction model that greatly simplifies the process of developing a NoSQL-based application. It scales horizontally with high availability and transparent load balancing even when dynamically adding new capacity.

Starting from the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 23ai release, the Oracle NoSQL Handler has been changed to use the Oracle NoSQL Java SDK to communicate with Oracle NoSQL. The Oracle NoSQL Java SDK supports both on-premise and OCI cloud instances of Oracle NoSQL. Make sure to read the documentation because connecting to on-premise versus OCI cloud instances of Oracle NoSQL both require specialized configuration parameters and possibly some setup.

For more information about Oracle NoSQL Java SDK, see [Oracle NoSQL SDK for Java](#).

10.2.33.2 On-Premise Connectivity

The Oracle NoSQL Java SDK requires that connectivity route through the Oracle NoSQL Database Proxy. The Oracle NoSQL Database Proxy is a separate process which enables the http/https interface of Oracle NoSQL. The Oracle NoSQL Java SDK uses the http/https interface. Oracle GoldenGate effectively communicates with the on-premise Oracle NoSQL instance through the Oracle NoSQL Database Proxy process.

For more information on the Oracle NoSQL Database Proxy including setup instructions, see [Connecting to the Oracle NoSQL Database On-premise](#).

Connectivity to the Oracle NoSQL Database Proxy requires mutual authentication whereby the client authenticates the server and the server authenticates the client.

- [Server Authentication](#)
- [Client Authentication](#)
- [Sample On-Premise Oracle NoSQL Configuration](#)

10.2.33.2.1 Server Authentication

Upon initial connection, the Oracle NoSQL Database Proxy process passes a certificate to the Oracle NoSQL Java SDK (Oracle NoSQL Handler). The Oracle NoSQL Java SDK then verifies the certificate against a certificate in a configured trust store. After the certificate received from the proxy has been verified against the trust store, the client has authenticated the server.

10.2.33.2.2 Client Authentication

Upon initial connection, the Oracle NoSQL Java SDK (Oracle NoSQL Handler) passes credentials (username and password) to the Oracle NoSQL Database Proxy. These credentials are used for the NoSQL On-Premise instance to client.

10.2.33.2.3 Sample On-Premise Oracle NoSQL Configuration

```
gg.handlerlist=nosql
gg.handler.nosql.type=nosql
gg.handler.nosql.nosqlURL=https://localhost:5555
gg.handler.nosql.ddlHandling=CREATE,ADD,DROP
gg.handler.nosql.interactiveMode=false
#Client Credentials
gg.handler.nosql.username={your username}
gg.handler.nosql.password={your password}
gg.handler.nosql.mode=op
# Set the gg.classpath to pick up the Oracle NoSQL Java SDK
gg.classpath=/path/to/the/SDK/*
# Set the -D options in the bootoptions to resolve the trust store location
and password
```

```
jvm.bootoptions=-Xmx512m -Xms32m -Djavax.net.ssl.trustStore=/usr/nosql/
kv-20.3.17/USER/security/driver.trust -
Djavax.net.ssl.trustStorePassword={your trust store password}
```

10.2.33.3 OCI Cloud Connectivity

Connectivity to an OCI Cloud instance of Oracle NoSQL is easier as it does not require the Oracle NoSQL Database Proxy required by the on-premise instance. Again, there is mutual authentication whereby the client authenticates the server and the server authenticates the client.

- [Server Authentication](#)
- [Client Authentication](#)
- [Sample Cloud Oracle NoSQL Configuration](#)
- [Sample OCI Configuration file](#)

10.2.33.3.1 Server Authentication

Upon initial connection, the Oracle NoSQL cloud instance passes a CA signed certificate to the client. The client then authenticates this CA signed certificate with the Certificate Authority. Once complete, the client has authenticated the server.

10.2.33.3.2 Client Authentication

Upon initial connection, the fingerprint, keyfile, and pass_phrase properties are used for the server to authenticate the client.

10.2.33.3.3 Sample Cloud Oracle NoSQL Configuration

```
gg.handlerlist=nosql
gg.handler.nosql.type=nosqlNoSQLSdkHandler
#gg.handler.nosql.type=nosql
gg.handler.nosql.ddlHandling=CREATE,ADD,DROP
gg.handler.nosql.interactiveMode=false
gg.handler.nosql.region=us-sanjose-1
gg.handler.nosql.configFilePath=/path/to/the/OCI/conf/file/nosql.conf
gg.handler.nosql.compartmentId=ocid1.compartment.oc1..aaaaaaaae2aedhka4jlb3h6z
hpaonaoktmg53adwkhwjflvv6hihz5cvwfeq
gg.handler.nosql.storageGb=10
gg.handler.nosql.readUnits=50
gg.handler.nosql.writeUnits=50
gg.handler.nosql.mode=op
# Set the gg.classpath to pick up the Oracle NoSQL Java SDK
gg.classpath=/path/to/the/SDK/*
```

10.2.33.3.4 Sample OCI Configuration file

```
[DEFAULT]
user=ocid1.user.oc1..aaaaaaaammf6u5h4wsmiuk52us5vnqhnnzyexkn56cqijlyo4vaa02jz
i3a
fingerprint=77:53:2c:e5:31:81:48:c3:3d:af:60:cf:e0:42:5c:7f
```

```
tenancy=ocid1.tenancy.oc1..aaaaaaaattuxbj75pnn3nksvzyidshdbrfmme flv4kkemajroz2  
thvca4kba  
region=us-sanjose-1  
key_file=/home/username/OracleNoSQL/lastname.firstname-04-13-18-51.pem  
openssl rsa -aes256 -in in.pem -out out.pem
```

tenancy

The Tenancy ID is displayed at the bottom of the Console page.

region

The region is displayed with the header session drop-down menu in the Console.

fingerprint

To generate the fingerprint, use the *How to Get the Key's Fingerprint* instructions at:
<https://docs.cloud.oracle.com/iaas/Content/API/Concepts/apisigningkey.htm>

key_file

You need to share the public and private key to establish a connection with Oracle Cloud Infrastructure. To generate the keys, use the *How to Generate an API Signing Key* at:
<https://docs.cloud.oracle.com/iaas/Content/API/Concepts/apisigningkey.htm>

pass_phrase

This is an optional property. It is used to configure the passphrase if the private key in the pem file is protected with a passphrase. The following openssl command can be used to take an unprotected private key pem file and add a passphrase.

The following command prompts the user for the passphrase:

```
openssl rsa -aes256 -in in.pem -out out.pem
```

For more information, see [Configuring Credentials for Oracle Cloud Infrastructure](#).

10.2.33.4 Oracle NoSQL Types

Oracle NoSQL provides a number of column data types and most of these data types are supported by the Oracle NoSQL Handler. A data type conversion from the column value in the trail file to the corresponding Java type representing the Oracle NoSQL column type in the Oracle NoSQL Handler is required.

The Oracle NoSQL Handler does not support Array, Map and Record data types by default. To support them, you can implement a custom data converter and override the default data type conversion logic to override it with your own custom logic to support your use case. Contact Oracle Support for guidance.

The following Oracle NoSQL data types are supported:

- Binary
- Boolean
- Double
- Integer
- Number
- String
- Timestamp

The following Oracle NoSQL data types are not supported:

- Array
- Map

10.2.33.5 Oracle NoSQL Handler Configuration

Properties	Require d/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.type</code>	Required	<code>nosql</code>	None	Selects the Oracle NoSQL Handler.
<code>gg.handler.name.interactiveMode</code>	Optional	<code>true</code> <code>false</code>	<code>true</code>	When set to <code>true</code> , the NoSQL handler will process one operation at a time. When set to <code>false</code> , the NoSQL Handler will process the batch perations at transaction commit. Batching has limitations. Batched operations must be separated by table and all batch operations for a table must have a common shared key(s).
<code>gg.handler.name.ddlHandling</code>	Optional	<code>CREATE</code> , <code>ADD</code> , <code>DROP</code> in any combination separated by a comma delimiter	None	Configure the Oracle NoSQL Handler for the DDL functionality to provide. Options include <code>CREATE</code> , <code>ADD</code> , and <code>DROP</code> . • When <code>CREATE</code> is enabled, the handler creates tables in Oracle NoSQL if a corresponding table does not exist. • When <code>ADD</code> is enabled, the handler adds columns that exist in the source table definition, but do not exist in the corresponding target Oracle NoSQL table definition. • When <code>DROP</code> is enabled, the handler drops columns that exist in the Oracle NoSQL table definition, but do not exist in the corresponding source table definition.
<code>gg.handler.name.retries</code>	Optional	Positive Integer	3	The number of retries on any read or write exception that the Oracle NoSQL Handler encounters.
<code>gg.handler.name.requestTimeout</code>	Optional	Positive Integer	30000	The maximum time in milliseconds for a NoSQL request to wait for a response. If the timeout is exceeded, the call is assumed to have failed.
<code>gg.handler.name.noSQLURL</code>	Optional	A valid URL including protocol	None	On-premise only. Used to set the connectivity URL for the NoSQL proxy instance.
<code>gg.handler.name.username</code>	Optional	String	None	On-premise only. Used to set the username for connectivity to an on-premise NoSQL instance through the NoSQL proxy process.
<code>gg.handler.name.password</code>	Optional	String	None	On-premise only. Used to set the password for connectivity to an on-premise NoSQL instance through the NoSQL proxy process.

Properties	Req uire d/ Opti onal	Legal Values	Default	Explanation
<code>gg.handler.name.compartmentId</code>	Optio nal	The OCID of an Oracle NoSQL compartment on OCI.	None	Cloud only. The OCID of an Oracle NoSQL cloud instance compartment on OCI.
<code>gg.handler.name.region</code>	Optio nal	Legal Oracle OCI region name.	None	Cloud only. The OCI region name of an Oracle NoSQL cloud instance.
<code>gg.handler.name.configFilePath</code>	Optio nal	A legal path and file name.	None	Cloud only. Set the path and file name of the config file containing the Oracle OCI information on the user, fingerprint, tenancy, region, and key-file.
<code>gg.handler.name.profile</code>	Optio nal	None	"DEFAULT"	Cloud only. Sets the named sub-section in the <code>gg.handler.name.configFilePath</code> . OCI config files can contain multiple entries and the naming specifies which entry to use.
<code>gg.handler.name.storageGb</code>	Optio nal	Positive Integer	10	Cloud only. Oracle NoSQL tables created in a cloud instance must be configured with a maximum storage size. This sets that configuration for tables created by the Oracle NoSQL Handler.
<code>gg.handler.name.readUnits</code>	Optio nal	Positive Integer	50	Cloud only. Oracle NoSQL tables created in an OCI cloud instance must be configured with read units which is the maximum read throughput. Each unit is 1KB per second.
<code>gg.handler.name.writeUnits</code>	Optio nal	Positive Integer	50	Cloud only. Oracle NoSQL tables created in an OCI cloud instance must be configured with write units which is the maximum write throughput. Each unit is 1KB per second.
<code>gg.handler.name.abendOnUnmappedColumns</code>	Optio nal	true false	true	Set to <code>true</code> if the desired behavior of the handler is to abend when a column is found in the source table but the column does not exist in the target NoSQL table. Set to <code>false</code> if the desired behavior is for the handler to ignore columns found in the source table for which no corresponding column exists in the target NoSQL table.
<code>gg.handler.name.dataConverterClasses</code>	Optio nal	The fully qualified data converter class name.	The default data converter.	The custom data converter can be implemented to override the default data conversion logic to support your specific use case. Must be included in the <code>gg.classpath</code> to be used.

Properties	Req uire d/ Opti onal	Legal Values	Default	Explanation
gg.handler.name. timestampPattern	Optio nal	A legal pattern for parsing timesta mps as they exist in the source trail file.	yyyy-MM- dd HH:mm:ss	This feature can be used to parse source field data into timestamps for timestamp target fields. The pattern needs to follow the Java convention for timestamp patterns and source data needs to conform to the pattern.
gg.handler.name. proxyServer	Optio nal	None	The proxy server host name.	Used to configure the forwarding proxy server host name for connectivity of on-premise Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) to Oracle Cloud Infrastructure (OCI) cloud instances of Oracle NoSQL. You must use at least version 5.2.27 of the Oracle NoSQL Java SDK.
gg.handler.name. proxyPort	Optio nal	80	Positive Integer	Used to configure the forwarding proxy server port number for connectivity of on-premise GG for DAA to OCI cloud instances of Oracle NoSQL. You must use at least version 5.2.27 of the Oracle NoSQL Java SDK.
gg.handler.name. proxyUsername	Optio nal	None	String	Used to configure the username of the forwarding proxy for connectivity of on-premise GG for DAA to OCI cloud instances of Oracle NoSQL if applicable. Most proxy servers do not require credentials. You must use at least version 5.2.27 of the Oracle NoSQL Java SDK.
gg.handler.name. proxyPassword	Optio nal	None	String	Used to configure the password of the forwarding proxy for connectivity of on-premise GG for DAA to OCI cloud instances of Oracle NoSQL if applicable. Most proxy servers do not require credentials. Must use at least version 5.2.27 of the Oracle NoSQL Java SDK.

10.2.33.6 Performance Considerations

When the NoSQL Handler is processing in interactive mode, operations are processed one at a time as they are received by the NoSQL Handler.

The NoSQL Handler will process in bulk mode if the following parameter is set.

```
gg.handler.name.interactiveMode=false
```

The NoSQL SDK allows bulk processing of operations for operations which meet the following criteria:

1. Operations must be for the same NoSQL table.
2. Operations must be in the same NoSQL shard (have the same shard key or shard key values).
3. Only one operation per row exists in the batch.

When interactive mode is set to false, the NoSQL handler group operations by table and shard key, and deduplicates operations for the same row.

An example of Deduplication: If there is an insert and an update for a row, then only the update operation is processed if the operations fall within the same transaction or replicat grouped transaction.

The NoSQL handler may provide better performance when interactive mode is set to false. However, for the interactive mode to provide better performance, operations need to be groupable by the above criteria. If operations are not groupable by the above criteria or if operations or bulk mode only provide grouping into very small batches, then bulk mode may not provide much or any improvement in performance.

10.2.33.7 Operation Processing Support

The Oracle NoSQL Handler moves operations to Oracle NoSQL using synchronous API. The insert, update, and delete operations are processed differently in Oracle NoSQL databases rather than in a traditional RDBMS:

The following explains how insert, update, and delete operations are interpreted by the handler depending on the mode of operation:

- insert: If the row does not exist in your database, then an insert operation is processed as an insert. If the row exists, then an insert operation is processed as an update.
- update: If a row does not exist in your database, then an update operation is processed as an insert. If the row exists, then an update operation is processed as update.
- delete: If the row does not exist in your database, then a delete operation has no effect. If the row exists, then a delete operation is processed as a delete.

The state of the data in Oracle NoSQL databases is idempotent. You can replay the source trail files or replay sections of the trail files. Ultimately, the state of an Oracle NoSQL database is the same regardless of the number of times the trail data was written into Oracle NoSQL.

Primary key values for a row in Oracle NoSQL databases are immutable. An update operation that changes any primary key value for a Oracle NoSQL row must be treated as a delete and insert. The Oracle NoSQL Handler can process update operations that result in the change of a primary key in an Oracle NoSQL database only as a delete and insert. To successfully process this operation, the source trail file must contain the complete before and after change data images for all columns.

10.2.33.8 Column Processing

You can configure the Oracle NoSQL Handler to add columns that exist in the source trail file table definition though are missing in the Oracle NoSQL table definition. The Oracle NoSQL Handler can accommodate metadata change events of adding a column. A reconciliation process occurs that reconciles the source table definition to the Oracle NoSQL table definition. When configured to add columns, any columns found in the source table definition that do not exist in the Oracle NoSQL table definition are added. The reconciliation process for a table occurs after application start up the first time an operation for the table is encountered. The reconciliation process reoccurs after a metadata change event on a source table, when the first operation for the source table is encountered after the change event.

Drop Column Functionality

Similar to adding, you can configure the Oracle NoSQL Handler to drop columns. The Oracle NoSQL Handler can accommodate metadata change events of dropping a column. A reconciliation process occurs that reconciles the source table definition to the Oracle NoSQL

table definition. When configured to drop columns, any columns found in the Oracle NoSQL table definition that are not in the source table definition are dropped.

 Caution

Dropping a column is potentially dangerous because it is permanently removing data from an Oracle NoSQL Database. Carefully consider your use case before configuring dropping.

Primary key columns cannot be dropped.

Column name changes are not handled well because there is no DDL-processing. The Oracle NoSQL Handler can handle any case change for the column name. A column name change event on the source database appears to the handler like dropping an existing column and adding a new column.

10.2.33.9 Table Check and Reconciliation Process

1. The Oracle NoSQL Handler interrogates the target Oracle NoSQL database for the table definition. If the table does not exist, the Oracle NoSQL Handler does one of two things. If `gg.handler.name.ddlHandling` includes `CREATE`, then a table is created in the database. Otherwise, the process abends and a message is logged that tells you the table that does not exist.
 2. If the table exists in the Oracle NoSQL database, then the Oracle NoSQL Handler performs a reconciliation between the table definition from the source trail file and the table definition in the database. This reconciliation process searches for columns that exist in the source table definition and not in the corresponding database table definition. If it locates columns fitting this criteria and the `gg.handler.name.ddlHandling` property includes `ADD`, then the Oracle NoSQL Handler alters the target table in the database to add the new columns. Otherwise the columns missing in the target will not be added. If the property `gg.handler.name.abendOnUnmappedColumns` is set to `true`, then the NoSQL Handler will abend. Else, if the configuration property `gg.handler.name.abendOnUnmappedColumns` is set to `false`, then the NoSQL Handler will continue the process and will not replicate data for the columns which exist in the source table and do not exist in the target NoSQL table.
 3. The reconciliation process searches for columns that exist in the target Oracle NoSQL and do not exist in the source table definition. If it locates columns fitting this criteria and the `gg.handler.name.ddlHandling` property includes `DROP`, then the Oracle NoSQL Handler alters the target table in Oracle NoSQL to drop these columns. Otherwise, those columns are ignored.
- [Full Image Data Requirements](#)

10.2.33.9.1 Full Image Data Requirements

In Oracle NoSQL, update operations perform a complete reinsertion of the data for the entire row. This Oracle NoSQL feature improves ingest performance, but in turn levies a critical requirement. Updates must include data for all columns, also known as full image updates. Partial image updates are not supported (updates with just the primary key information and data for the columns that changed). Using the Oracle NoSQL Handler with partial image update information results in incomplete data in the target NoSQL table.

10.2.33.10 Oracle NoSQL SDK Dependencies

The maven coordinates are as follows:

Maven groupId: com.oracle.nosql.sdk

Maven artifactId: nosqldriver

Version: 5.2.27

- [Oracle NoSQL SDK Dependencies 5.2.27](#)

10.2.33.10.1 Oracle NoSQL SDK Dependencies 5.2.27

```
bcpkix-jdk15on-1.68.jar  
bcprov-jdk15on-1.68.jar  
jackson-core-2.12.1.jar  
netty-buffer-4.1.63.Final.jar  
netty-codec-4.1.63.Final.jar  
netty-codec-http-4.1.63.Final.jar  
netty-codec-socks-4.1.63.Final.jar  
netty-common-4.1.63.Final.jar  
netty-handler-4.1.63.Final.jar  
netty-handler-proxy-4.1.63.Final.jar  
netty-resolver-4.1.63.Final.jar  
netty-transport-4.1.63.Final.jar  
nosqldriver-5.2.27.jar
```

10.2.34 OCI Autonomous Data Warehouse

Oracle Autonomous Data Warehouse (ADW) is a fully managed database tuned and optimized for data warehouse workloads with the market-leading performance of Oracle Database.

- [Detailed Functionality](#)
The ADW Event handler is used as a downstream Event handler connected to the output of the OCI Object Storage Event handler. The OCI Event handler loads files generated by the File Writer Handler into Oracle OCI Object storage. All the SQL operations are performed in batches providing better throughput.
- [ADW Database Credential to Access OCI ObjectStore File](#)
- [ADW Database User Privileges](#)
ADW databases come with a predefined database role named DWR0LE. If the ADW 'admin' user is not being used, then the database user needs to be granted the role DWR0LE.
- [Unsupported Operations/ Limitations](#)
- [Troubleshooting and Diagnostics](#)
- [Classpath](#)
ADW apply relies on the upstream File Writer handler and the OCI Event handler. Include the required jars needed to run the OCI Event handler in gg.classpath.
- [Configuration](#)
- [Restrictions on Database Object Names](#)

10.2.34.1 Detailed Functionality

The ADW Event handler is used as a downstream Event handler connected to the output of the OCI Object Storage Event handler. The OCI Event handler loads files generated by the File Writer Handler into Oracle OCI Object storage. All the SQL operations are performed in batches providing better throughput.

10.2.34.2 ADW Database Credential to Access OCI ObjectStore File

To access the OCI ObjectStore File:

1. A PL/SQL procedure needs to be run to create a credential to access Oracle Cloud Infrastructure (OCI) Object store files.
2. An OCI authentication token needs to be generated under User settings from the OCI console. For example:

```
BEGIN DBMS_CLOUD.create_credential
(   credential_name =>
    'OGGBD-CREDENTIAL',   username => 'oci-user',   password =>
    'oci-user');
END;
/
```

3. The credential name can be configured using the following property:
gg.eventhandler.adw.objectStoreCredential. For example:
gg.eventhandler.adw.objectStoreCredential=OGGBD-CREDENTIAL.

10.2.34.3 ADW Database User Privileges

ADW databases come with a predefined database role named DWROLE. If the ADW 'admin' user is not being used, then the database user needs to be granted the role DWROLE.

This role provides the privileges required for data warehouse operations. For example, the following command grants DWROLE to the user dbuser-1:

```
GRANT DWROLE TO dbuser-1;
```

Note

Ensure that you do not use Oracle-created database user ggadmin for ADW replication, because this user lacks the INHERIT privilege.

10.2.34.4 Unsupported Operations/ Limitations

- DDL changes are not supported.
- Replication of Oracle Object data types are not supported.
- If the GoldenGate trail is generated by Oracle Integrated capture, then for the UPDATE operations on the source LOB column, only the changed portion of the LOB is written to the trail file. Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA)

Autonomous Data Warehouse (ADW) apply doesn't support replication of partial LOB columns in the trail file.

- **ADW Event Handler and External Table Schema Constraints:** There is a known limitation in the ADW event handler where external tables cannot be created within their own dedicated schema. This impacts replication configurations involving multiple distinct target schemas, for example, eb1, eb2, and eb3. As a result, each unique target schema requires its own dedicated replicat group to ensure proper handling and isolation.

Current Configuration Example: The following mapping rules illustrate a multi-schema replication setup:

```
MAP eb1.* , TARGET eb1.* ;
MAP eb2.* , TARGET eb2.* ;
MAP eb3.* , TARGET eb3.* ;
```

In this scenario, due to the ADW limitation, you must configure separate replicat groups for each schema, for example, one for eb1 and one for eb2, rather than using a single replicat group for all mappings.

Workaround: Use individual replicat groups per target schema to bypass the schema creation restriction for external tables.

Note

OCI Autonomous Data Warehouse does not currently support multi-schema in Oracle GoldenGate replication due to dependency in external table creation under the admin user.

10.2.34.5 Troubleshooting and Diagnostics

- **Connectivity Issues to ADW**
 - Validate JDBC connection URL, user name and password.
 - Check if http/https proxy is enabled. See ADW proxy configuration: [Prepare for Oracle Call Interface \(OCI\), ODBC, and JDBC OCI Connections](#) in *Using Oracle Autonomous Data Warehouse on Shared Exadata Infrastructure*.
- **DDL not applied on the target table: The ADW handler will ignore DDL.**
- **Target table existence:** It is expected that the ADW target table exists before starting the apply process. Target tables need to be designed with appropriate primary keys, indexes and partitions. Approximations based on the column metadata in the trail file may not be always correct. Therefore, replicat will ABEND if the target table is missing.
- **Diagnostic throughput information on the apply process is logged into the handler log file.**

For example:

```
File Writer finalized 29525834 records
      (rate: 31714) (start time: 2020-02-10 01:25:32.000579) (end time:
2020-02-10
      01:41:03.000606).
```

In this sample log message:

- This message provides details about the end-end throughput of File Writer handler and the downstream event handlers (OCI Event handler and ADW event handler).
- The throughput rate also takes into account the wait-times incurred before rolling over files.
- The throughput rate also takes into account the time taken by the OCI event handler and the ADW event handler to process operations.
- The above examples indicates that 29525834 operations were finalized at the rate of 31714 operations per second between start time: [2020-02-10 01:25:32.000579] and end time: [2020-02-10 01:41:03.000606].

Example:

```
INFO 2019-10-01 00:36:49.000490 [pool-8-thread-1] - Begin DWH Apply stage
and load statistics
*****START*****

INFO 2019-10-01 00:36:49.000490 [pool-8-thread-1] - Time spent for staging
process [2074 ms]
INFO 2019-10-01 00:36:49.000490 [pool-8-thread-1] - Time spent for merge
process [992550 ms]
INFO 2019-10-01 00:36:49.000490 [pool-8-thread-1] - [31195516] operations
processed, rate[31,364]operations/sec.

INFO 2019-10-01 00:36:49.000490 [pool-8-thread-1] - End DWH Apply stage
and load statistics
*****END*****

INFO 2019-10-01 00:37:18.000230 [pool-6-thread-1] - Begin OCI Event
handler upload statistics
*****START*****

INFO 2019-10-01 00:37:18.000230 [pool-6-thread-1] - Time spent loading
files into ObjectStore [71789 ms]
INFO 2019-10-01 00:37:18.000230 [pool-6-thread-1] - [31195516] operations
processed, rate[434,545] operations/sec.
INFO 2019-10-01 00:37:18.000230 [pool-6-thread-1] - End OCI Event handler
upload statistics
*****END*****
```

In this example:

ADW Event handler throughput:

- In the above log message, the statistics for the ADW event handler is reported as *DWH Apply stage and load statistics*. ADW is classified as a Data Ware House (DWH), and therefore, this name.
- Here 31195516 operations from the source trail file were applied to ADW database at the rate of 31364 operations per second.
- ADW uses stage and merge. The time spent on staging is 2074 milliseconds and the time spent on executing merge SQL is 992550 milliseconds.

OCI Event handler throughput:

- In the above log message, the statistics for the OCI event handler is reported as *OCI Event handler upload statistics*.

- Here 31195516 operations from the source trail file were uploaded to the OCI object store at the rate of 434545 operations per second.
- **Errors due to ADW credential missing grants to read OCI object store files:**
 - A SQL exception indicating authorization failure is logged in the handler log file. For example:

```
java.sql.SQLException: ORA-20401:
Authorization failed for URI -
https://objectstorage.us-ashburn-1.oraclecloud.com/n/some_namespace/b/
some_bucket/o/ADMIN.NLS_AllTypes/ADMIN.NLS_AllTypes_2019-12-16_11-44-01.237.avro
```

- **Errors in file format/column data:**

In case the ADW Event handler is unable to read data from the external staging table due to column data errors, the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) handler log file provides diagnostic information to debug the issue.

The following details are available in the log file:

- JOB ID
- SID
- SERIAL #
- ROWS_LOADED
- START_TIME
- UPDATE_TIME
- STATUS
- TABLE_NAME
- OWNER_NAME
- FILE_URI_LIST
- LOGFILE_TABLE
- BADFILE_TABLE

The contents of the LOGFILE_TABLE and BADFILE_TABLE should indicate the specific record and the column(s) in the record which have error and the cause of the error. This information is also queried automatically by the ADW Event handler and logged into the OGGBD FW handler log file. Based on the root cause of the error, customer can take action. In many cases, customers would have to modify the target table definition based on the source column data types and restart replicat. In other cases, customers may also want to modify the mapping in the replicat prm file. For this, Oracle recommends that they re-position replicat to start from the beginning.

- **Any other SQL Errors:**

In case there are any errors while executing any SQL, the entire SQL statement along with the bind parameter values are logged into the OGGBD handler log file.
- **Co-existence of the components:**

The location/region of the machine where replicat process is running, OCI Objects storage bucket region and the ADW region would impact the overall throughput of the apply process. Data flow is as follows: **GoldenGate** → **OCI Object store** → **ADW**. For best throughput, the components need to be located as close as possible.
- **Debugging row count mismatch on the target table**

For better throughput, ADW event handler does not validate the row counts modified on the target table. We can enable row count matching by using the Java System property:

`disable.row.count.validation`. To enable row count validation, provide this property in the `jvm.bootoptions` as follows: `jvm.bootoptions=-Xmx512m -Xms32m -Djava.class.path=.:ggjava/ggjava.jar:./dirprm -Ddisable.row.count.validation=false`

- **Replicat ABEND due to partial LOB records in the trail file:**
GG for DAA ADW apply does not support replication of partial LOB. The trail file needs to be regenerated by Oracle Integrated capture using `TRANLOGOPTIONS FETCHPARTIALLOB` option in the extract parameter file.
- **Throughput gain with uncompressed UPDATE trails:**
If the source trail files contain the full image (all the column values of the respective table) of the row being updated, then you can include the JVM boot option - `Dcompressed.update=false` in the configuration property `jvm.bootoptions`.

For certain workloads and ADW instance shapes, this configuration may provide a better throughput. You may need to test the throughput gain on your environment.

10.2.34.6 Classpath

ADW apply relies on the upstream File Writer handler and the OCI Event handler. Include the required jars needed to run the OCI Event handler in `gg.classpath`.

ADW Event handler uses the Oracle JDBC driver and its dependencies. The Autonomous Data Warehouse JDBC driver and other required dependencies are packaged with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

For example: `gg.classpath=./oci-java-sdk/lib/*:./oci-java-sdk/third-party/lib/*`

10.2.34.7 Configuration

- [Automatic Configuration](#)
Autonomous Data Warehouse (ADW) replication involves configuring of multiple components, such as file writer handler, OCI event handler and ADW event handler.
- [File Writer Handler Configuration](#)
File writer handler name is pre-set to the value `adw`. The following is an example to edit a property of file writer handler: `gg.handler.adw.pathMappingTemplate=./dirout`
- [OCI Event Handler Configuration](#)
OCI event handler name is pre-set to the value 'oci'.
- [ADW Event Handler Configuration](#)
ADW event handler name is pre-set to the value `adw`.
- [INSERTALLRECORDS Support](#)
- [In-Memory Operation Aggregation](#)
- [End-to-End Configuration](#)
- [Compressed Update Handling](#)

10.2.34.7.1 Automatic Configuration

Autonomous Data Warehouse (ADW) replication involves configuring of multiple components, such as file writer handler, OCI event handler and ADW event handler.

The Automatic Configuration functionality helps to auto configure these components so that the user configuration is minimal. The properties modified by auto configuration will also be logged in the handler log file.

To enable auto configuration to replicate to ADW target we need to set the parameter

```
gg.target=adw
```

```
gg.target
```

Required

Legal Value: adw

Default: None

Explanation: Enables replication to ADW target

When replicating to ADW target, customization of OCI event handler name and ADW event handler name is not allowed.

10.2.34.7.2 File Writer Handler Configuration

File writer handler name is pre-set to the value adw. The following is an example to edit a property of file writer handler: `gg.handler.adw.pathMappingTemplate=./dirout`

10.2.34.7.3 OCI Event Handler Configuration

OCI event handler name is pre-set to the value 'oci'.

The following is an example to edit a property of the OCI event handler:

```
gg.eventhandler.oci.profile=DEFAULT
```

10.2.34.7.4 ADW Event Handler Configuration

ADW event handler name is pre-set to the value adw.

The following are the ADW event handler configurations:

Property	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.adw.connectionURL</code>	Required	ADW	None	Sets the ADW JDBC connection URL. Example: <code>jdbc:oracle:thin:@adw20190410ns_medium?TNS_ADMIN=/home/sanav/projects/adw/wallet</code>
<code>gg.eventhandler.adw.UserName</code>	Required	JDBC User name	None	Sets the ADW database user name.
<code>gg.eventhandler.adw.Password</code>	Required	JDBC Password	None	Sets the ADW database password.
<code>gg.eventhandler.adw.maxStatements</code>	Optional	Integer value between 1 to 250.	The default value is 250.	Use this parameter to control the number of prepared SQL statements that can be used.
<code>gg.eventhandler.adw.maxConnections</code>	Optional	Integer value.	10	Use this parameter to control the number of concurrent JDBC database connections to the target ADW database.
<code>gg.eventhandler.adw.dropStagingTablesOnShutdown</code>	Optional	true false	false	If set to true, the temporary staging tables created by the ADW event handler is dropped on replicat graceful stop.

Property	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler.adw.objectStoreCredential	Required	A database credential name.	None	ADW Database credential to access OCI object-store files.
gg.initialLoad	Optional	true false	false	If set to true, initial load mode is enabled. See INSERTALLRECORDS Support .
gg.compressed.update	Optional	true or false	true	If set to true, then this indicates that the source trail files contain compressed update operations. If set to false, then the source trail files are expected to contain uncompressed update operations.
gg.eventhandler.adw.connectionRetries	Optional	Integer Value	3	Specifies the number of times connections to the target data warehouse will be retried.
gg.eventhandler.adw.connectionRetryIntervalSecs	Optional	Integer Value	30	Specifies the delay in seconds between connection retry attempts.

Property	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.adw. fileRollInterval</code>	Optional	The default unit of measure is milliseconds. You can stipulate ms, s, m, h to signify milliseconds, seconds, minutes, or hours respectively. Examples of legal values include 10000, 10000 ms, 10s, 10m, or 1.5h. Values of 0 or less indicate that file rolling on time is turned off.	3m (three minutes)	The parameter determines how often the data will be merged into ADW. Use with caution, the higher this value is the more data will need to be stored in the memory of the Replicat process.

Note

Use the parameter with caution. Increasing its default value (3m) will increase the amount of data stored in the internal memory of the Replicat. This can cause out of memory errors and stop the Replicat if it runs out of memory.

Note

Starting with the 23ai release, the `gg.aggregate.operations.flush.interval` property is deprecated and no longer supported. For more information, see [In-Memory Operation Aggregation](#).

Property	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler.adw.deleteInsert	Optional	true or false	false	If set to true, Replicat will merge records using SQL DELETE+INSERT statements instead of SQL MERGE statement.
Note Applicable only if gg.compressed.update is set to false.				
gg.validate.key update	Optional	true or false	false	If set to true, Replicat will validate key update operations (optype 115) and correct to normal update if no key values have changed. Compressed key update operations do not qualify for merge.

10.2.34.7.5 INSERTALLRECORDS Support

Stage and merge targets supports INSERTALLRECORDS parameter.

See [INSERTALLRECORDS](#) in *Reference for Oracle GoldenGate*. Set the INSERTALLRECORDS parameter in the Replicat parameter file (.prm). Set the INSERTALLRECORDS parameter in the Replicat parameter file (.prm)

Setting this property directs the Replicat process to use bulk insert operations to load operation data into the target table.

You can tune the batch size of bulk inserts using the File writer property

gg.handler.adw.maxFileSize. The default value is set to 1GB. The frequency of bulk inserts can be tuned using the File writer property gg.handler.adw.fileRollInterval, the default value is set to 3m (three minutes).

To process initial load trail files, set the INSERTALLRECORDS parameter in the Replicat parameter file (.prm). Setting this property directs the Replicat process to use bulk insert operations to load operation data into the target table.

You can tune the batch size of bulk inserts using the File Writer property

gg.handler.adw.maxFileSize. The default value is set to 1GB. The frequency of bulk inserts can be tuned using the File Writer property gg.handler.adw.fileRollInterval, the default value is set to 3m (three minutes).

10.2.34.7.6 In-Memory Operation Aggregation

- Operation records are aggregated in-memory by default.
- The gg.aggregate.operations.flush.interval property has been deprecated and is no longer supported. If gg.aggregate.operations.flush.interval is used in GG for DAA 26ai, then replicat will run; but add a warning to log file about the property being deprecated and not supported.
To control the time window for aggregation, use gg.handler.adw.fileRollInterval property. By default, it is set to 3 minutes. Longer intervals will increase latency, and may increase memory usage. Shorter intervals will increase overhead in Oracle GoldenGate and the target database.

- Operation aggregation in-memory requires additional JVM memory configuration.

10.2.34.7.7 End-to-End Configuration

The following is an end-end configuration example which uses auto configuration for FW handler, OCI and ADW Event handlers. The sample properties file is available at the following location:

- **In an Oracle GoldenGate Classic install:** <oggbd_install_dir>/AdapterExamples/big-data/adw-via-oci/adw.props.
- **In an Oracle GoldenGate Microservices install:** <oggbd_install_dir>/opt/AdapterExamples/big-data/adw-via-oci/adw.props.

```
# Configuration to load GoldenGate trail operation records
# into Autonomous Data Warehouse (ADW) by chaining
# File writer handler -> OCI Event handler -> ADW Event handler.
# Note: Recommended to only edit the configuration marked as TODO
gg.target=adw
##The OCI Event handler
# TODO: Edit the OCI config file path.
gg.eventhandler.oci.configFilePath=<path/to/oci/config>
# TODO: Edit the OCI profile name.
gg.eventhandler.oci.profile=DEFAULT
# TODO: Edit the OCI namespace.
gg.eventhandler.oci.namespace=<OCI namespace>
# TODO: Edit the OCI region.
gg.eventhandler.oci.region=<oci-region>
# TODO: Edit the OCI compartment identifier.
gg.eventhandler.oci.compartmentID=<OCI compartment id>
gg.eventhandler.oci.pathMappingTemplate=${fullyQualifiedTableName}
# TODO: Edit the OCI bucket name.
gg.eventhandler.oci.bucketMappingTemplate=<ogg-bucket>
##The ADW Event Handler
# TODO: Edit the ADW JDBC connectionURL
gg.eventhandler.adw.connectionURL=jdbc:oracle:thin:@adw20190410ns_medium?TNS_ADMIN=/
path/to/ /adw/wallet
# TODO: Edit the ADW JDBC user
gg.eventhandler.adw.UserName=<db user>
# TODO: Edit the ADW JDBC password
gg.eventhandler.adw.Password=<db password>
# TODO: Edit the ADW Credential that can access the OCI Object Store.
gg.eventhandler.adw.objectStoreCredential=<ADW Object Store credential>
# TODO:Set the classpath to include OCI Java SDK.
gg.classpath=./oci-java-sdk/lib/*:./oci-java-sdk/third-party/lib/*
#TODO: Edit to provide sufficient memory (at least 8GB).
jvm.bootoptions=-Xmx8g -Xms8g
```

10.2.34.7.8 Compressed Update Handling

A compressed update record contains values for the key columns and the modified columns.

An uncompressed update record contains values for all the columns.

Oracle GoldenGate trails may contain compressed or uncompressed update records. The default extract configuration writes compressed updates to the trails.

The parameter `gg.compressed.update` can be set to true or false to indicate compressed/uncompressed update records.

- [MERGE Statement with Uncompressed Updates](#)

10.2.34.7.8.1 MERGE Statement with Uncompressed Updates

In some use cases, if the trail contains uncompressed update records, then the MERGE SQL statement can be optimized for better performance by setting `gg.compressed.update=false`.

10.2.34.8 Restrictions on Database Object Names

- **Catalog, Schema, and Table Names:** Cannot contain period (`.`), asterisk (`*`), question mark (`?`), double quote (`"`), backslash (`\`), or control characters (ASCII 0-31).
- **Specific limitation for GGDA 21c and previous releases:** Supports only names that comply with the Avro specification for identifier names: must start with `[A-Z a-z _]`, followed by `[A-Za-z0-9 _]`.
- Additionally, all object names are also limited by the supported list of characters for the respective target.

10.2.35 OCI Object Storage

The Oracle Cloud Infrastructure (OCI) Event Handler is used to load files generated by the File Writer Handler into an Oracle Cloud Infrastructure Object Store.

The Oracle Cloud Infrastructure Event Handler is used to load files generated by the [Flat Files](#) into an Oracle Cloud Infrastructure Object Store. This topic describes how to use the OCI Event Handler.

- [Overview](#)
- [Detailing the Functionality](#)
- [Configuration](#)
- [Configuring Credentials for Oracle Cloud Infrastructure](#)
- [Troubleshooting](#)
- [OCI Dependencies](#)

10.2.35.1 Overview

The Oracle Cloud Infrastructure Object Storage service is an internet-scale, high-performance storage platform that offers reliable and cost-efficient data durability. The Object Storage service can store an unlimited amount of unstructured data of any content type, including analytic data and rich content, like images and videos, see https://cloud.oracle.com/en_US/cloud-infrastructure.

You can use any format handler that the File Writer Handler supports.

10.2.35.2 Detailing the Functionality

The Oracle Cloud Infrastructure Event Handler requires the Oracle Cloud Infrastructure Java software development kit (SDK) to transfer files to Oracle Cloud Infrastructure Object Storage. Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) does not include the Oracle Cloud Infrastructure Java SDK, see <https://docs.cloud.oracle.com/iaas/Content/API/Concepts/sdkconfig.htm>.

You must download the Oracle Cloud Infrastructure Java SDK at:

<https://docs.us-phoenix-1.oraclecloud.com/Content/API/SDKDocs/javasdk.htm>

Extract the JAR files to a permanent directory. There are two directories required by the handler, the JAR library directory that has Oracle Cloud Infrastructure SDK JAR and a third-party JAR library. Both directories must be in the `gg.classpath`.

Specify the `gg.classpath` environment variable to include the JAR files of the Oracle Cloud Infrastructure Java SDK.

Example

```
gg.classpath=/usr/var/oci/lib/*:/usr/var/oci/third-party/lib/*
```

Setting of the proxy server settings requires additional dependency libraries identified by the following Maven coordinates:

Group ID: `com.oracle.oci.sdk`

Artifact ID: `oci-java-sdk-addons-apache`

The best way to get all of the dependencies is to use the Dependency Downloading utility scripts. The OCI script downloads both the OCI Java SDK and the Apache Addons libraries.

For more information on this dependency, see [OCI Documentation - README](#).

10.2.35.3 Configuration

You configure the Oracle Cloud Infrastructure Event Handler operation using the properties file. These properties are located in the Java Adapter properties file (and not in the Replicat properties file).

The Oracle Cloud Infrastructure Event Handler works only in conjunction with the File Writer Handler.

To enable the selection of the Oracle Cloud Infrastructure Event Handler, you must first configure the handler type by specifying `gg.eventhandler.name.type=oci` and the other Oracle Cloud Infrastructure properties as follows:

Table 10-40 Oracle Cloud Infrastructure Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.type</code>	Required	<code>oci</code>	None	Selects the Oracle Cloud Infrastructure Event Handler.
<code>gg.eventhandler.name.contentType</code>	Optional	Valid content type value which is used to indicate the media type of the resource.	<code>application/octet-stream</code>	The content type of the object.
<code>gg.eventhandler.name.contentEncoding</code>	Optional	Valid values indicate which encoding to be applied.	<code>utf-8</code>	The content encoding of the object.

Table 10-40 (Cont.) Oracle Cloud Infrastructure Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.contentLanguage</code>	Optional	Valid language intended for the audience.	en	The content language of the object.
<code>gg.eventhandler.name.configFilePath</code>	Optional	Path to the event handler config file.	None	The configuration file name and location. If <code>gg.eventhandler.name.configFilePath</code> is not set, then the following authentication parameters are required: <code>gg.eventhandler.name.userId</code> <code>gg.eventhandler.name.tenancyID</code> <code>gg.eventhandler.name.region</code> <code>gg.eventhandler.name.privateKeyFile</code> <code>gg.eventhandler.name.publicKeyFingerprint</code> These parameters take precedence over <code>gg.eventhandler.name.configFilePath</code>.
<code>gg.eventhandler.name.userId</code>	Optional	Valid user ID	None	OCID of the user calling the API. To get the value, see (Required Keys and OCIDs)https://docs.oracle.com/en-us/iaas/Content/API/Concepts/apisigningkey.htm#Required_Keys_and_OCIDs. Example: <code>ocid1.user.oc1..<unique_ID></code> (shortened for brevity)
<code>gg.eventhandler.name.tenancyId</code>	Optional	Valid tenancy ID	None	OCID of your tenancy. To get the value, see (Required Keys and OCIDs)https://docs.oracle.com/en-us/iaas/Content/API/Concepts/apisigningkey.htm#Required_Keys_and_OCIDs in <i>Oracle Cloud Infrastructure</i> documentation. Example: <code>ocid1.tenancy.oc1..<unique_ID></code>

Table 10-40 (Cont.) Oracle Cloud Infrastructure Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler .name.privateKeyFile	Optional	A valid path to the file	None	Full path and filename of the private key.
 Note The key pair must be in PEM format. For more information about generating a key pair in PEM format, see (Required Keys and OCIDs) https://docs.oracle.com/en-us/iaas/Content/API/Concepts/apisigningkey.htm#Required_Keys_and_OCIDs in <i>Oracle Cloud Infrastructure</i> documentation. Example: /home/opc/.oci/oci_api_key.pem				
gg.eventhandler .name.publicKeyFingerprint	Optional	String	None	Fingerprint for the public key that was added to this user. To get the value, see (Required Keys and OCIDs) https://docs.oracle.com/en-us/iaas/Content/API/Concepts/apisigningkey.htm#Required_Keys_and_OCIDs in <i>Oracle Cloud Infrastructure</i> documentation.
gg.eventhandler .name.profile	Required	Valid string representing the profile name.	DEFAULT	In the Oracle Cloud Infrastructure config file, the entries are identified by the profile name. The default profile is DEFAULT. You can have an additional profile like ADMIN_USER. Any value that isn't explicitly defined for the ADMIN_USER profile (or any other profiles that you add to the config file) is inherited from the DEFAULT profile.
gg.eventhandler .name.region	Required	Oracle Cloud Infrastructure region	None	Oracle Cloud Infrastructure Servers and Data is hosted in a region and is a localized geographic area. The valid Region Identifiers are listed at Oracle Cloud Infrastructure Documentation - Regions and Availability Domains .

Table 10-40 (Cont.) Oracle Cloud Infrastructure Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.compartmentID</code>	Required	Valid compartment id.	None	A compartment is a logical container to organize Oracle Cloud Infrastructure resources. The compartmentID is listed in Bucket Details while using the Oracle Cloud Infrastructure Console.
<code>gg.eventhandler.name.pathMappingTemplate</code>	Required	A string with resolvable keywords and constants used to dynamically generate the path in the Oracle Cloud Infrastructure bucket to write the file.	None	Use keywords interlaced with constants to dynamically generate unique Oracle Cloud Infrastructure path names at runtime. See Template Keywords .
<code>gg.eventhandler.name.fileNameMappingTemplate</code>	Optional	A string with resolvable keywords and constants used to dynamically generate the Oracle Cloud Infrastructure file name at runtime.	None	Use resolvable keywords and constants to dynamically generate the Oracle Cloud Infrastructure data file name at runtime. If not set, the upstream file name is used. See Template Keywords .
<code>gg.eventhandler.name.bucketMappingTemplate</code>	Required	A string with resolvable keywords and constants used to dynamically generate the path in the Oracle Cloud Infrastructure bucket to write the file.	None	Use resolvable keywords and constants used to dynamically generate the Oracle Cloud Infrastructure bucket name at runtime. The event handler attempts to create the Oracle Cloud Infrastructure bucket if it does not exist. See Template Keywords .

Table 10-40 (Cont.) Oracle Cloud Infrastructure Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.finalizeAction</code>	Optional	none delete	None	Set to none to leave the Oracle Cloud Infrastructure data file in place on the finalize action. Set to delete if you want to delete the Oracle Cloud Infrastructure data file with the finalize action.
<code>gg.eventhandler.name.eventHandler</code>	Optional	A unique string identifier cross referencing a child event handler.	No event handler is configured.	Sets the event handler that is invoked on the file roll event. Event handlers can do file roll event actions like loading files to S3, converting to Parquet or ORC format, loading files to HDFS, loading files to Oracle Cloud Infrastructure Storage Classic, or loading file to Oracle Cloud Infrastructure.
<code>gg.eventhandler.name.proxyServer</code>	Optional	The host name of your proxy server.	None	Set to the host name of the proxy server if OCI connectivity requires routing through a proxy server.
<code>gg.eventhandler.name.proxyPort</code>	Optional	The port number of the proxy server.	None	Set to the port number of the proxy server if OCI connectivity requires routing through a proxy server.
<code>gg.eventhandler.name.proxyProtocol</code>	Optional	HTTP HTTPS	HTTP	Sets the proxy protocol connection to the proxy server for additional level of security. The majority of proxy servers support HTTP. Only set this if the proxy server supports HTTPS and HTTPS is required.
<code>gg.eventhandler.name.proxyUsername</code>	Optional	The username for the proxy server.	None	Sets the username for connectivity to the proxy server if credentials are required. Most proxy servers do not require credentials.
<code>gg.eventhandler.name.proxyPassword</code>	Optional	The password for the proxy server.	None	Sets the password for connectivity to the proxy server if credentials are required. Most proxy servers do not require credentials.
<code>gg.handler.name.SSEKey</code>	Optional	A legal Base64 encoded OCI server side encryption key.	None	Allows you to control the encryption of data files loaded to OCI. OCI encrypts by default. This property allows an additional level of control by supporting encryption with a specific key. That key must also be used to decrypt data files.

Sample Configuration

```

gg.eventhandler.oci.type=oci
gg.eventhandler.oci.configFilePath=~/.oci/config
gg.eventhandler.oci.profile=DEFAULT
gg.eventhandler.oci.namespace=dwcsdemo
gg.eventhandler.oci.region=us-ashburn-1
gg.eventhandler.oci.compartmentID=ocid1.compartment.oc1..aaaaaaaajdg6iblwqglyqpegf6kwdaais
2gyx3guspboa7fsi72tfihz2wrba

```

```
gg.eventhandler.oci.pathMappingTemplate=${schemaName}
gg.eventhandler.oci.bucketMappingTemplate=${schemaName}
gg.eventhandler.oci.fileNameMappingTemplate=${tableName}_${currentTimestamp}.txt
gg.eventhandler.oci.finalizeAction=NONE
goldengate.userexit.writers=javawriter
```

- [Automatic Configuration](#)

10.2.35.3.1 Automatic Configuration

OCI Object storage replication involves configuring multiple components, such as the File Writer Handler, formatter, and the target OCI Object Storage Event Handler.

The Automatic Configuration functionality helps you to auto configure these components so that the manual configuration is minimal.

The properties modified by auto-configuration is also logged in the handler log file.

To enable auto configuration to replicate to the OCI Object Storage target, set the parameter `gg.target=oci`.

- [File Writer Handler Configuration](#)
- [Formatter Configuration](#)

10.2.35.3.1.1 File Writer Handler Configuration

The File Writer Handler name is pre set to the value `oci`.

You can add or edit a property of the File Writer Handler. For example:

```
gg.handler.oci.pathMappingTemplate=./dirout
```

10.2.35.3.1.2 Formatter Configuration

The json row formatter is set by default.

You can add or edit a property of the formatter. For example:

```
gg.handler.oci.format=json_row
```

10.2.35.4 Configuring Credentials for Oracle Cloud Infrastructure

Basic configuration information like user credentials and tenancy Oracle Cloud IDs (OCIDs) of Oracle Cloud Infrastructure is required for the Java SDKs to work, see <https://docs.cloud.oracle.com/iaas/Content/General/Concepts/identifiers.htm>.

The ideal configuration file include keys user, fingerprint, key_file, tenancy, and region with their respective values. The default configuration file name and location is `~/.oci/config`.

Create the config file as follows:

1. Create a directory called `.oci` in the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) home directory
2. Create a text file and name it `config`.
3. Obtain the values for these properties:

user

- a. Login to the Oracle Cloud Infrastructure Console <https://console.us-ashburn-1.oraclecloud.com>.
- b. Click **Username**.
- c. Click **User Settings**.

The User's OCID is displayed and is the value for the key user.

tenancy

The Tenancy ID is displayed at the bottom of the Console page.

region

The region is displayed with the header session drop-down menu in the Console.

fingerprint

To generate the fingerprint, use the *How to Get the Key's Fingerprint* instructions at:

<https://docs.cloud.oracle.com/iaas/Content/API/Concepts/apisigningkey.htm>

key_file

You need to share the public and private key to establish a connection with Oracle Cloud Infrastructure. To generate the keys, use the *How to Generate an API Signing Key*:

<https://docs.cloud.oracle.com/iaas/Content/API/Concepts/apisigningkey.htm>

pass_phrase

This is an optional property. It is used to configure the passphrase if the private key in the pem file is protected with a passphrase. The following openssl command can be used to take an unprotected private key pem file and add a passphrase.

The following command prompts the user for the passphrase:

```
openssl rsa -aes256 -in in.pem -out out.pem
```

Sample Configuration File

```
user=ocid1.user.oc1..aaaaaaaat5nvwcn5j6aqzqedqw3rynjq
fingerprint=20:3b:97:13::4e:c5:3a:34
key_file=~/.oci/oci_api_key.pem
tenancy=ocid1.tenancy.oc1..aaaaaaaaba3pv6wkcr44h25vqstifs
```

10.2.35.5 Troubleshooting

Connectivity Issues

If the OCI Event Handler is unable to connect to the OCI object storage when running on premise, it's likely your connectivity to the public internet is protected by a proxy server. Proxy servers act a gateway between the private network of a company and the public internet. Contact your network administrator to get the URL of your proxy server.

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) connectivity to OCI can be routed through a proxy server by setting the following configuration properties:

```
gg.eventhandler.name.proxyServer={insert your proxy server name}
gg.eventhandler.name.proxyPort={insert your proxy server port number}
```

ClassNotFoundException Error

The most common initial error is an incorrect classpath that does not include all the required client libraries so results in a `ClassNotFoundException` error. Specify the `gg.classpath` variable to include all of the required JAR files for the Oracle Cloud Infrastructure Java SDK, see [Detailing the Functionality](#).

Duplicate records after Replicat Recovery

OCI Object Storage handler replication uses File Writer Handler and OCI Object Storage handler in the replicat. Oracle GoldenGate prioritizes no data loss and guarantees no data loss in case of failures by at least once semantics in OCI Object Storage (json, csv, delimitedtext, avro_orc, parquet) delivery. In the cases if replicat runs fine and normally shut down, then exactly once is supported. In case of failures (because of network failures), there are various reason that can lead into duplicates in recovery.

Two cases where duplicates can occur:

1. If data is written and a failure occurs between when the data is written, and when the checkpoint is moved. Then upon restart the replicat backs up to the previous checkpoint and data can unfortunately be replayed.
2. The rolling of the data files occurs based on customer configured triggers. Trigger can be file size, time, inactivity, or time of day. The rolling does not necessarily happen on a transaction commit boundary. The trigger causes writing to the current file to complete, the post processing transformation and movement complete, and any state on that file is deleted. If a replicat abend occurs in between when the rolling is processed and when the checkpoint is moved, then upon restart, it can again replay those messages.

If you observe duplicate records in case of OCI Object Storage replicat recovery, then it is an expected behavior. If you observe duplicates while replicat is running fine, then file a support ticket.

10.2.35.6 OCI Dependencies

The maven coordinates for OCI are as follows:

Maven groupId: com.oracle.oci.sdk

Maven artifactId: oci-java-sdk-full

Version: 1.34.0

The following are the Apache add-ons to which, support routing through a proxy server:

Maven groupId: com.oracle.oci.sdk

Maven artifactId: oci-java-sdk-addons-apache

Version: 1.34.0

- [OCI 1.34.0](#)

10.2.35.6.1 OCI 1.34.0

accessors-smart-1.2.jar
aopalliance-repackaged-2.6.1.jar
asm-5.0.4.jar
bcpkix-jdk15on-1.68.jar
bcprov-jdk15on-1.68.jar

checker-qual-3.5.0.jar
commons-codec-1.15.jar
commons-io-2.8.0.jar
commons-lang3-3.8.1.jar
commons-logging-1.2.jar
error_prone_annotations-2.3.4.jar
failureaccess-1.0.1.jar
guava-30.1-jre.jar
hk2-api-2.6.1.jar
hk2-locator-2.6.1.jar
hk2-utils-2.6.1.jar
httpclient-4.5.13.jar
httpcore-4.4.13.jar
j2objc-annotations-1.3.jar
jackson-annotations-2.12.0.jar
jackson-core-2.12.0.jar
jackson-databind-2.12.0.jar
jackson-datatype-jdk8-2.12.0.jar
jackson-datatype-jsr310-2.12.0.jar
jackson-module-jaxb-annotations-2.10.1.jar
jakarta.activation-api-1.2.1.jar
jakarta.annotation-api-1.3.5.jar
jakarta.inject-2.6.1.jar
jakarta.ws.rs-api-2.1.6.jar
jakarta.xml.bind-api-2.3.2.jar
javassist-3.25.0-GA.jar
jcip-annotations-1.0-1.jar
jersey-apache-connector-2.32.jar
jersey-client-2.32.jar
jersey-common-2.32.jar
jersey-entity-filtering-2.32.jar
jersey-hk2-2.32.jar
jersey-media-json-jackson-2.32.jar
json-smart-2.3.jar
jsr305-3.0.2.jar
listenablefuture-9999.0-empty-to-avoid-conflict-with-guava.jar
nimbus-jose-jwt-8.5.jar
oci-java-sdk-addons-apache-1.34.0.jar
oci-java-sdk-analytics-1.34.0.jar
oci-java-sdk-announcementsservice-1.34.0.jar
oci-java-sdk-apigateway-1.34.0.jar
oci-java-sdk-apmcontrolplane-1.34.0.jar
oci-java-sdk-apmsynthetics-1.34.0.jar
oci-java-sdk-apmtraces-1.34.0.jar
oci-java-sdk-applicationmigration-1.34.0.jar
oci-java-sdk-artifacts-1.34.0.jar
oci-java-sdk-audit-1.34.0.jar
oci-java-sdk-autoscaling-1.34.0.jar
oci-java-sdk-bds-1.34.0.jar
oci-java-sdk-blockchain-1.34.0.jar
oci-java-sdk-budget-1.34.0.jar
oci-java-sdk-cims-1.34.0.jar
oci-java-sdk-circuitbreaker-1.34.0.jar
oci-java-sdk-cloudguard-1.34.0.jar
oci-java-sdk-common-1.34.0.jar
oci-java-sdk-computeinstanceagent-1.34.0.jar
oci-java-sdk-containerengine-1.34.0.jar
oci-java-sdk-core-1.34.0.jar
oci-java-sdk-database-1.34.0.jar
oci-java-sdk-databasemanagement-1.34.0.jar
oci-java-sdk-datacatalog-1.34.0.jar
oci-java-sdk-dataflow-1.34.0.jar

```
oci-java-sdk-dataintegration-1.34.0.jar
oci-java-sdk-datasafe-1.34.0.jar
oci-java-sdk-datascience-1.34.0.jar
oci-java-sdk-dns-1.34.0.jar
oci-java-sdk-dts-1.34.0.jar
oci-java-sdk-email-1.34.0.jar
oci-java-sdk-events-1.34.0.jar
oci-java-sdk-filestorage-1.34.0.jar
oci-java-sdk-full-1.34.0.jar
oci-java-sdk-functions-1.34.0.jar
oci-java-sdk-goldengate-1.34.0.jar
oci-java-sdk-healthchecks-1.34.0.jar
oci-java-sdk-identity-1.34.0.jar
oci-java-sdk-integration-1.34.0.jar
oci-java-sdk-keymanagement-1.34.0.jar
oci-java-sdk-limits-1.34.0.jar
oci-java-sdk-loadbalancer-1.34.0.jar
oci-java-sdk-loganalytics-1.34.0.jar
oci-java-sdk-logging-1.34.0.jar
oci-java-sdk-loggingingestion-1.34.0.jar
oci-java-sdk-loggingsearch-1.34.0.jar
oci-java-sdk-managementagent-1.34.0.jar
oci-java-sdk-managementdashboard-1.34.0.jar
oci-java-sdk-marketplace-1.34.0.jar
oci-java-sdk-monitoring-1.34.0.jar
oci-java-sdk-mysql-1.34.0.jar
oci-java-sdk-networkloadbalancer-1.34.0.jar
oci-java-sdk-nosql-1.34.0.jar
oci-java-sdk-objectstorage-1.34.0.jar
oci-java-sdk-objectstorage-extensions-1.34.0.jar
oci-java-sdk-objectstorage-generated-1.34.0.jar
oci-java-sdk-oce-1.34.0.jar
tbcampbe: oci-java-sdk-ocvp-1.34.0.jar
oci-java-sdk-oda-1.34.0.jar
oci-java-sdk-ons-1.34.0.jar
oci-java-sdk-opsi-1.34.0.jar
oci-java-sdk-optimizer-1.34.0.jar
oci-java-sdk-osmanagement-1.34.0.jar
oci-java-sdk-resourcemanager-1.34.0.jar
oci-java-sdk-resourcearch-1.34.0.jar
oci-java-sdk-rover-1.34.0.jar
oci-java-sdk-sch-1.34.0.jar
oci-java-sdk-secrets-1.34.0.jar
oci-java-sdk-streaming-1.34.0.jar
oci-java-sdk-tenantmanagercontrolplane-1.34.0.jar
oci-java-sdk-usageapi-1.34.0.jar
oci-java-sdk-vault-1.34.0.jar
oci-java-sdk-waas-1.34.0.jar
oci-java-sdk-workrequests-1.34.0.jar
osgi-resource-locator-1.0.3.jar
resilience4j-circuitbreaker-1.2.0.jar
resilience4j-core-1.2.0.jar
slf4j-api-1.7.29.jar
vavr-0.10.0.jar
vavr-match-0.10.0.jar
```

10.2.36 Redis

Redis is an in-memory data structure store which supports optional durability. Redis is simply a key/value data store where a unique key identifies the data structure stored. The value is the data structure that is stored.

The Redis Handler supports the replication of change data capture to Redis and the storage of that data in three different data structures: Hash Maps, Streams, JSONs.

- [Data Structures Supported by the Redis Handler](#)
- [Redis Handler Configuration Properties](#)
- [Security](#)
- [Authentication Using Credentials](#)
- [SSL Basic Auth](#)
- [SSL Mutual Auth](#)
- [Redis Handler Dependencies](#)
The Redis Handler uses the Jedis client libraries to connect to the Redis server.
- [Redis Handler Client Dependencies](#)
The Redis Handler uses the Jedis client to connect to Redis.

10.2.36.1 Data Structures Supported by the Redis Handler

- [Hash Maps](#)
- [Streams](#)
- [JSONs](#)

10.2.36.1.1 Hash Maps

This is the most common user use case. The key is a unique identifier for the table and row of the data which is being pushed to Redis. The data structure stored at each key location is a hash map. The key in the hash map is the column name and the value is the column value.

Behavior on Inserts, Updates, and Deletes

The source trail file will contain insert, update, and delete operations for which the data can be pushed into Redis. The Redis Handler will process inserts, updates, and deletes as follows:

Inserts – The Redis Handler will create a new key in Redis the value of which is a hash map for which the hash map key is the column name and the hash map value is the column value.

Updates – The Redis Handler will update an existing hash map structure in Redis. The existing hash map will be updated with the column names and values from the update operation processed. Because hash map data is updated and not replace, full image updates are not required.

Primary Key Updates – The Redis Handler will move the old key to the new key name alone with the data structure, then an update will be performed on the hash map.

Deletes – The Redis Handler will delete the key and its corresponding data structure from Redis.

Handling of Null Values

Redis hash maps cannot store null as a value. A Redis hash map must have a non-null value. The default behavior is to omit columns with a null value from the generated hash map. If an update changes a column value from a non-null value to a null value, then the column key and value is removed from the hash map.

Users may wish to propagate null values to Redis. But, because Redis hash maps cannot store null values, a representative value will need to be configured to be propagated instead. This is configured by setting the following two parameters:

```
gg.handler.redis.omitNullValues=false
gg.handler.redis.nullValueRepresentation=null
```

The user will need to designate some value as null. But the following are legal too.

In this case the null value representation is an empty string or "".

```
gg.handler.redis.nullValueRepresentation=CData[]
```

In this case the null value representation is set to a tab.

```
gg.handler.redis.nullValueRepresentation=CDATA[\\t]
```

Support for Binary Values

The default functionality is to push all data into Redis hash maps as Java strings. Binary values must be converted to Base64 to be represented as a Java String. Consequently, binary values will be represented as Base64. Alternatively, users can push bytes into Redis hash maps to retain the original bytes values by setting the following configuration property.

```
qq.handler.redis.dataType=bytes
```

Example hash map data in Redis:

```
127.0.0.1:6379> hgetall TCUSTOMER:JANE
1) "optype"
2) "I"
3) "CITY"
4) "DENVER"
5) "primarykeycolumns"
6) "CUST_CODE"
7) "STATE"
8) "CO"
9) "CUST_CODE"
10) "JANE"
11) "position"
12) "0000000000000000002126"
13) "NAME"
14) "ROCKY FLYER INC."
```

Example Configuration

```
gg.handlerlist=redis
gg.handler.redis.type=redis
gg.handler.redis.hostPortList= localhost:6379
gg.handler.redis.createIndexes=true
gg.handler.redis.mode=op
gg.handler.redis.metacolumnsTemplate=${position},${optype},${primarykeycolumns}
```

10.2.36.1.2 Streams

Redis streams are analogous to the Kafka topics. The Redis key is the stream name. The value of the stream are the individual messages pushed to the Redis stream. Individual messages are identified by a timestamp and offset of when the message was pushed to Redis. The value of each individual message is a hash map for which the key is the column name and value is the column value.

Behavior on Inserts, Updates, and Deletes

Each and every operation and its associated data is propagated to Redis Streams. Therefore, every operation will show up as a new message in Redis Streams.

Handling of Null Values

Redis streams stores hash maps as the value for each message. A Redis hash map cannot store null as a value. Null values work exactly as they do in hash maps functionality.

Support for Binary Values

The default functionality is to push all data into Redis hash maps as Java strings. Binary values must be converted to Base64 to be represented as a Java String. Consequently, binary values will be represented as Base64. Alternatively, users can push bytes into Redis hash maps to retain the original bytes values by setting the following configuration property.

```
gg.handler.redis.dataType=bytes
```

Stream data appears in Redis as follows:

```
127.0.0.1:6379> xread STREAMS TCUSTMER 0-0
```

```
1) 1) "TCUSTMER"
    2) 1) 1) "1664399290398-0"
        2) 1) "optype"
            2) "I"
            3) "CITY"
            4) "SEATTLE"
            5) "primarykeycolumns"
            6) "CUST_CODE"
            7) "STATE"
            8) "WA"
            9) "CUST_CODE"
            10) "WILL"
            11) "position"
            12) "00000000000000001956"
            13) "NAME"
            14) "BG SOFTWARE CO."

2) 1) "1664399290398-1"
    2) 1) "optype"
        2) "I"
        3) "CITY"
        4) "DENVER"
        5) "primarykeycolumns"
        6) "CUST_CODE"
        7) "STATE"
        8) "CO"
        9) "CUST_CODE"
        10) "JANE"
        11) "position"
        12) "00000000000000002126"
        13) "NAME"
        14) "ROCKY FLYER INC."
```

Example Configuration

```
gg.handlerlist=redis
gg.handler.redis.type=redis
gg.handler.redis.hostportlist=localhost:6379
gg.handler.redis.mode=op
```

```
gg.handler.redis.integrationType=streams
gg.handler.redis.metacolumnsTemplate=${position},${optype},${primarykeycolumns}
```

10.2.36.1.3 JSONs

The key is a unique identifier for the table and row of the data which is being pushed to Redis. The value is a JSON object. The keys in the JSON object are the column names while the values in the JSON object are the column values.

The source trail file will contain inserts update and delete operations for which the data can be pushed into Redis. The Redis Handler will process inserts, updates, and deletes as follows:

Inserts – The Redis Handler will create a new JSON at the key.

Updates – The Redis Handler will replace the JSON at the given key with the new JSON reflecting the data of update. Because the JSON is replaced, full image updates are recommended in the source trail file.

Deletes – The key in Redis along with its corresponding JSON data structure are deleted.

Handling of Null Values

The JSON specification supports null values as JSON null. Therefore, null values in the data will be propagated as JSON null. Null value replacement is not supported since the JSON specification supports null values. Neither `gg.handler.redis.omitNullValues` nor `gg.handler.redis.nullValueRepresentation` configuration properties have any effect when the Redis Handler is configured to send JSONs. JSON per the specification is represented as follows: `"fieldname": null`

Support for Binary Values

Per the JSON specification, binary values are represented as Base64. Therefore, all binary values will be converted and propagated as Base64. Setting the property `gg.handler.redis.dataType` has no effect. JSONs will generally appear in Redis as follows:

```
127.0.0.1:6379> JSON.GET  
TCUSTOMER:JANE{"position":"0000000000000000002126","optype":"","primarykey":  
ycolumns":["CUST_CODE"],"CUST_CODE":"JANE","NAME":"ROCKY FLYER  
INC.","CITY":"DENVER","STATE":"CO"}"
```

Example Configuration:

```
gg.handlerlist=redis
gg.handler.redis.type=redis
gg.handler.redis.hostportlist=localhost:6379
gg.handler.redis.mode=op
gg.handler.redis.integrationType=jsons
gg.handler.redis.createIndexes=true
gg.handler.redis.metacolumnsTemplate=${position},${optype},${primarykeycolumns}
```

10.2.36.2 Redis Handler Configuration Properties

Table 10-41 Redis Handler Configuration Properties

Properties	Required/Optional	Legal Values	Default	Explanation
gg.handlerlist= name	Required	Any String	none	Provides the name for the Redis Handler.

Table 10-41 (Cont.) Redis Handler Configuration Properties

Properties	Required/Optional	Legal Values	Default	Explanation
gg.handler.name.type	Required	redis	none	Selects the Redis Handler.
gg.handler.name.mode	Optional	op tx	op	The default is recommended. In op mode, operations are processed as received. In tx mode, operations are cached and processed at transaction commit. The tx mode is slower and creates a larger memory footprint.
gg.handler.name.integrationType	Optional	hashmaps streams jsons	hashmaps	Sets the integration type for Redis. Select hashmaps and the data will be pushed into Redis as hashmaps. Select streams and data will be pushed into Redis streams. Select jsons and the data will be pushed into Redis as JSONs.
gg.handler.name.keyMappingTemplate	Optional	string bytes	string	Only valid for hashmap and streams integration types. Controls if string data or byte data is pushed to Redis. If string is selected, all binary data will be pushed to Redis Base64 encoded. If bytes is selected, binary data is pushed to Redis without conversion.
gg.handler.name.keyMappingTemplate	Optional	Any combination of string and templating keywords.	For hashmaps and jsons: \${tableName}:\${primaryKeys} For streams: \${tableName}	Redis is a key value data store. The resolved value of this template determines the key for an operation.

Table 10-41 (Cont.) Redis Handler Configuration Properties

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.handler.name.createIndexes</code>	Optional	true false	true	Will automatically create an index for each replicated table for the following integration types: hashmaps jsons User can delete these indexes or create additional indexes. Information on created indexes is logged to the replicat <replicat name>.log file.
<code>gg.handler.name.omitNullValues</code>	Optional	true false	true	Null values cannot be stored as values in a Redis hashmap structure. Both the integration types hashmaps and streams store hashmaps. By default, if a column value is null it cannot be replicated to Redis. By default, if a column value is changed to null, it has to be removed from a hashmap. Setting this to false will replicate a configured value representing a null value to Redis.
<code>gg.handler.name.nullValueRepresentation</code>	Optional	Any String	"" (empty string)	Only valid if integration type is hashmaps or streams. Only valid if <code>gg.handler.name.omitNullValues</code> is set to false. This configured value here is the value that will be replicated to Redis instead of a null.

Table 10-41 (Cont.) Redis Handler Configuration Properties

Properties	Required/Optional	Legal Values	Default	Explanation
gg.handler.name .metaColumnsTemplate	Optional	Any string of comma separated metacolumn keywords.	none	This can be configured to select one or more metacolumns to be added to the output to Redis. See Metacolumn Keywords .
gg.handler.name .insertOpKey	Optional	Any string	"I"	This is the value of the operation type for inserts which is replicated if the metacolumn \$ {optype} is configured.
gg.handler.name .updateOpKey	Optional	Any sting	"U"	This is the value of the operation type for updates which is replicated if the metacolumn \$ {optype} is configured.
gg.handler.name .deleteOpKey	Optional	Any string	"D"	This is the value of the operation type for deletes which is replicated if the metacolumn \$ {optype} is configured.
gg.handler.name .truncateOpKey	Optional	Any string	"T"	This is the value of the operation type for truncate which is replicated if the metacolumn \$ {optype} is configured.
gg.handler.name .maxStreamLength	Optional	Positive Integer	0	Sets the maximum length of streams. If more messages are pushed to a stream than this value, then the oldest messages will be deleted so that the maximum stream size is enforced. The default value is 0 which means no limit on the maximum stream length.

Table 10-41 (Cont.) Redis Handler Configuration Properties

Properties	Required/Optional	Legal Values	Default	Explanation
gg.handler.name.username	Optional	Any string	None	Used to set the username, if required, for connectivity to Redis.
gg.handler.name.password	Optional	Any string	None	Used to set the password, if required, for connectivity to Redis.
gg.handler.name.timeout	Optional	integer	15000	Property to set the both the connection and socket timeouts in milliseconds.
gg.handler.name.enableSSL	Optional	true false	false	Set to true if connecting to a Redis that has been SSL enabled. SSL can be basic auth (certificate passes from server to client) or mutual auth (certificate passes from server to client and then a certificate passes from client to the server). Basic auth is generally combined with use of credentials (username and password) so that both sides of the connection can authenticate the other. SSL provides encryption of in flight messages.

10.2.36.3 Security

Connectivity to Redis can be secured in multiple ways. It is the Redis server which is configured for, and thereby selects, the type of security. The Redis Handler, which is the Redis client, must be configured to match the security of the server.

Redis server – connection listener – This is the Redis application.

Redis client – connection caller – This is the Oracle GoldenGate Redis Handler.

Check with your Redis administrator as to what security has been configured on the Redis server. Then, configure the Redis Handler to follow the security configuration of the Redis server.

10.2.36.4 Authentication Using Credentials

This is a simple security that requires the Redis client-provided credentials (username and password) for the Redis server to authenticate the Redis client. This security does not provide any encryption of inflight messages.

```
gg.handler.name.username=<username>  
gg.handler.name.password=<password>
```

10.2.36.5 SSL Basic Auth

In this use case the Redis server passes a certificate to the Redis client. This allows the client to authenticate the server. The client passes credentials to the server, which allows the Redis server to authenticate the client. This connection is SSL and provides encryption of inflight messages.

```
gg.handler.name.enableSSL=true  
gg.handler.name.username=<username>  
gg.handler.name.password=<password>
```

If the Redis server passes an unsigned certificate to the Redis client, then the Redis Handler will need to be configured with a truststore. If the Redis server passes a certificate signed by a Certificate Authority, then a truststore is not required.

Note

Security Recommendation: Oracle recommends storing credentials in Oracle GoldenGate Credential Store, Oracle Cloud Vault, or another enterprise credential management solution instead of storing plaintext passwords in configuration files.

To configure a truststore on the Redis Handler:

```
jvm.bootoptions=-Djavax.net.ssl.trustStore=<absolute path to truststore> -  
Djavax.net.ssl.trustStorePassword=<truststore password>
```

10.2.36.6 SSL Mutual Auth

In this use case the Redis server passes a certificate to the Redis client. This allows the client to authenticate the server. The Redis client then passes a certificate to the Redis server. This allows the server to authenticate the Redis client. This connection is SSL and provides encryption of inflight messages.

```
gg.handler.name.enableSSL=true
```


Typically with this setup, the Redis client will need both a truststore and a keystore. The configuration is as follows:

To configure a truststore on the Redis Handler:

```
jvm.bootoptions=-Djavax.net.ssl.keyStore=<absolute path to keystore> -  
Djavax.net.ssl.keyStorePassword=<keystore password> -  
Djavax.net.ssl.trustStore=<absolute path to truststore> -  
Djavax.net.ssl.trustStorePassword=<truststore password>
```

10.2.36.7 Redis Handler Dependencies

The Redis Handler uses the Jedis client libraries to connect to the Redis server.

The following is a link to Jedis: <https://github.com/redis/jedis>

The Jedis libraries do not ship with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) and will need to be obtained and then the `gg.classpath` configuration property will need to be configured to resolved the Jedis client. The dependency downloader utility which ships with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) can be used to download Jedis. The Redis Handler was developed using Jedis 4.2.3. The following shows example configuration of the `gg.classpath`:
`gg.classpath=/OGGBDinstall/DependencyDownloader/dependencies/jedis_4.2.3/*`

10.2.36.8 Redis Handler Client Dependencies

The Redis Handler uses the Jedis client to connect to Redis.

Group ID: redis.clients

Artifact ID: jedis

- [jedis 4.2.3](#)

10.2.36.8.1 jedis 4.2.3

commons-pool2-2.11.1.jar

gson-2.8.9.jar

jedis-4.2.3.jar

json-20211205.jar

slf4j-api-1.7.32.jar

10.2.37 Snowflake

Snowflake is a serverless data warehouse that runs on any of the following cloud providers: Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure. Oracle GoldenGate provides two handlers to load data into Snowflake:

- Snowflake Stage and Merge
- Snowflake Streaming Handler

The following table summarizes the differences between the two handlers:

Table 10-42 Differences between Stage and Merge, and Streaming Handlers

Details	Stage and Merge	Streaming Handler
Workload	Any	Inserts-only
Latency	Micro-batch	Real-time
Resources	Customer-managed virtual warehouse	Snowflake-managed serverless compute
API	JDBC+SQL commands (PUT/COPY INTO/MERGE)	Snowpipe Streaming
Data load	Requires intermediary loading step from Int/Ext Stage.	Direct loading into Snowflake

Topics:

- [Snowflake Stage and Merge Handler](#)
- [Snowflake Streaming Handler](#)
- [Supported Snowflake Date Datatypes](#)
- [Snowflake Iceberg Tables](#)

10.2.37.1 Snowflake Stage and Merge Handler

Topics:

- [Overview](#)
- [Detailed Functionality](#)
- [Configuration](#)
- [Troubleshooting and Diagnostics](#)
- [Restrictions on Database Object Names](#)

10.2.37.1.1 Overview

Snowflake is a serverless data warehouse that runs on any of the following cloud providers: Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure.

The Snowflake Event Handler is used to replicate data into Snowflake.

10.2.37.1.2 Detailed Functionality

Replication to Snowflake uses the stage and merge data flow.

- The change data from the Oracle GoldenGate trails is staged in micro-batches at a temporary staging location (internal or external stage).
- The staged records are then merged into the Snowflake target tables using a merge SQL statement.

This topic contains the following:

- [Staging Location](#)
- [Database User Privileges](#)
- [Prerequisites](#)

10.2.37.1.2.1 Staging Location

The change data records from the Oracle GoldenGate trail files are formatted into Avro OCF (Object Container Format) and are then uploaded to the staging location.

Change data can be staged in one of the following object stores:

- Snowflake internal stage
- Snowflake external stage
 - AWS Simple Storage Service (S3)
 - Azure Data Lake Storage (ADLS) Gen2
 - Google Cloud Storage (GCS)

10.2.37.1.2.2 Database User Privileges

The database user used for replicating into Snowflake has to be granted the following privileges:

- INSERT, UPDATE, DELETE, and TRUNCATE on the target tables.
- CREATE and DROP on Snowflake named stage and external stage.
- If using external stage (S3, ADLS, GCS), CREATE, ALTER, and DROP external table.

10.2.37.1.2.3 Prerequisites

- You must have Amazon Web Services, Google Cloud Platform, or Azure cloud accounts set up if you intend to use any of the external stage locations such as, S3, ADLS Gen2, or GCS.
- Snowflake JDBC driver
- If you want to maintain the source operation types in the target Snowflake tables while using Snowflake Streaming Handler, use the COLMAP type. For example:

```
"MAP QASOURCE.TCUSTORD, TARGET QASOURCE.TCUSTORD, COLMAP (USEDEFAULTS,  
op_type = @GETENV('GGHEADER', 'OPTYPE'));"
```

- If you do not want to maintain the source operation types in the target Snowflake tables while using Snowflake Streaming Handler, all the source operations will be ingested as the INSERT type to target Snowflake tables.

10.2.37.1.3 Configuration

The configuration of the Snowflake replication properties is stored in the Replicat properties file.

Note

Ensure to specify the path to the properties file in the parameter file only when using Coordinated Replicat. Add the following line to the parameter file:

```
TARGETDB LIBFILE libggjava.so SET property=<parameter file directory>/  
<properties file name>
```

- [Automatic Configuration](#)
- [Snowflake Storage Integration](#)
- [Classpath Configuration](#)
- [Proxy Configuration](#)
- [INSERTALLRECORDS Support](#)
- [Snowflake Key Pair Authentication](#)
- [Mapping Source JSON/XML to Snowflake VARIANT](#)
- [Operation Aggregation](#)
- [Compressed Update Handling](#)
- [End-to-End Configuration](#)
- [Table Mapping](#)

10.2.37.1.3.1 Automatic Configuration

Snowflake replication involves configuring multiple components, such as the File Writer Handler, S3 or HDFS or GCS Event Handler, and the target Snowflake Event Handler.

The Automatic Configuration functionality helps you to auto-configure these components so that the manual configuration is minimal.

The properties modified by auto-configuration is also logged in the handler log file.

To enable auto-configuration to replicate to the Snowflake target, set the parameter `gg.target=snowflake`.

The parameter `gg.stage` determines the staging location. If `gg.stage` is unset, then Snowflake internal stage is used.

If `gg.stage` is set to one of them - `s3`, `abs`, or `gcs`, then AWS S3, ADLS Gen2, or GCS are respectively used as the staging locations.

The JDBC Metadata provider is also automatically enabled to retrieve target table metadata from Snowflake.

- [File Writer Handler Configuration](#)
- [S3 Handler Configuration](#)
- [HDFS Event Handler Configuration](#)
- [Google Cloud Storage Event Handler Configuration](#)
- [Snowflake Event Handler Configuration](#)

10.2.37.1.3.1.1 File Writer Handler Configuration

The File Writer Handler name is pre-set to the value `snowflake` and its properties are automatically set to the required values for Snowflake.

You can add or edit a property of the File Writer Handler. For example:

```
gg.handler.snowflake.pathMappingTemplate=./dirout
```

10.2.37.1.3.1.2 S3 Handler Configuration

The S3 Event Handler name is pre-set to the value `s3` and must be configured to match your S3 configuration.

The following is an example of editing a property of the S3 Event Handler:

```
gg.eventhandler.s3.bucketMappingTemplate=bucket1
```

For more information, see [Amazon S3](#).

10.2.37.1.3.1.3 HDFS Event Handler Configuration

The Hadoop Distributed File System (HDFS) Event Handler name is pre-set to the value `hdfs` and it is auto-configured to write to HDFS.

Ensure that the Hadoop configuration file `core-site.xml` is configured to write data files to the respective container in the Azure Data Lake Storage (ADLS) Gen2 storage account. For more information, see [Azure Data Lake Gen2 using Hadoop Client and ABFS](#).

The following is an example of editing a property of the HDFS Event handler:

```
gg.eventhandler.hdfs.finalizeAction=delete
```

10.2.37.1.3.1.4 Google Cloud Storage Event Handler Configuration

The Google Cloud Storage (GCS) Event Handler name is pre-set to the value `gcs` and must be configured to match your GCS configuration.

The following is an example of editing a GCS Event Handler property:

```
gg.eventhandler.gcs.bucketMappingTemplate=bucket1
```

10.2.37.1.3.1.5 Snowflake Event Handler Configuration

The Snowflake Event Handler name is pre-set to the value `snowflake`.

The following are configuration properties available for the Snowflake Event handler, the required ones must be changed to match your Snowflake configuration:

Table 10-43 Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.snowflake.connectionURL	Required	Supported connection URL. For example, jdbc:snowflake://<account_name>.snowflakecomputing.com/?warehouse=<warehouse-name>&db=<database-name>	None	JDBC URL to connect to Snowflake. Snowflake account name, warehouse and database must be set in the JDBC URL. The warehouse can be set using warehouse=<warehouse name>, database can set using db=<db name>. In some cases for authorization, a role should be set using role=<rolename>
gg.eventhandler.snowflake.UserName	Required	Supported database user name string.	None	Snowflake database user.
gg.eventhandler.snowflake.Password	Required	Supported database password string.	None	Snowflake database password.
gg.eventhandler.snowflake.storageIntegration	Optional	Storage integration name.	None	This parameter is required when using an external stage such as ADLS Gen2 or GCS or S3. This is the credential for Snowflake data warehouse to access the respective Object store files. For more information, see Snowflake Storage Integration .
gg.eventhandler.snowflake.maxConnections	Optional	Integer value	10	Use this parameter to control the number of concurrent JDBC database connections to the target Snowflake database.

Table 10-43 (Cont.) Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .snowflake.drop StagingTablesOn Shutdown	Optional	true false	false	If set to true, the temporary staging tables created by Oracle GoldenGate are dropped on replicat graceful stop.

Table 10-43 (Cont.) Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation
gg.handler.snowflake.fileRollInterval	Optional	The default unit of measure is milliseconds. You can stipulate ms, s, m, h to signify milliseconds, seconds, minutes, or hours respectively. Examples of legal values include 10000, 10000ms, 10s, 10m, or 1.5h. Values of 0 or less indicate that file rolling on time is turned off.	3m (three minutes)	The parameter determines how often the data will be merged into Snowflake. Use with caution, the higher this value is the more data will need to be stored in the memory of the Replicat process.

Note
Use the parameter name to reference the parameter with caution. In case of a syntax error, the parameter name will be used as is.

Properties	Required/Optional	Legal Values	Default	Explanation

Properties	Required/Optional	Legal Values	Default	Explanation

ng its default value (3m) will increase the amount of data stored

Table 10-43 (Cont.) Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation

Table 10-43 (Cont.) Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation

ry
e
r
r
o
r
s
a
n
d
s
t
o
p
t
h
e
R
e
p
l
i
c
a
t
i
f
i
t
r
u
n
s
o
u
t
o
f
m
e
m
o
r
y
.



Table 10-43 (Cont.) Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation

t
e
S
t
a
r
t
i
n
g
w
i
t
h
t
h
e
2
3
a
i
r
e
l
e
a
s
e
,
t
h
e
g
g
.
a
g
g
r
e
g
a
t
e
.
o
p
e
r
a
t
i

Table 10-43 (Cont.) Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation

ons
s
.
f
l
u
s
h
.
i
n
t
e
r
v
a
l
p
r
o
p
e
r
t
y
i
s
d
e
p
r
e
c
a
t
e
d
a
n
d
n
o
l
o
n
g
e
r
s
u
p
p

Table 10-43 (Cont.) Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation

Oracle GoldenGate for Distributed Applications and Analytics
May 29, 2026
Page 588 of 771

Table 10-43 (Cont.) Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .snowflake.putS QLThreads	Optional	Integer Value	4	Specifies the number of threads ('PARALLEL` clause) to use for uploading files using PUT SQL. This is only relevant when Snowflake internal stage (named stage) is used.
gg.eventhandler .snowflake.putS QLAutoCompress	Optional	true false	false	Specifies whether Snowflake uses gzip to compress files (AUTO_COMPRESS clause) during upload using PUT SQL. true: Files are compressed (if they are not already compressed). false: Files are not compressed (which means, the files are uploaded as is). This is only relevant when Snowflake internal stage (named stage) is used.
gg.validate.key update	Optional	true or false	false	If set to true, Replicat will validate key update operations (optype 115) and correct to normal update if no key values have changed. Compressed key update operations do not qualify for merge.

Table 10-43 (Cont.) Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.snowflake.useCopyForInitialLoad	Optional	true or false	true	If set to true, then COPY SQL statement will be used during initial load. If set to false, then INSERT SQL statement will be used during initial load.
gg.compressed.update	Optional	true or false	true	If set the true, then this indicates that the source trail files contain compressed update operations. If set to false, then the source trail files are expected to contain uncompressed update operations.
gg.eventhandler.snowflake.connectionRetries	Optional	Integer Value	3	Specifies the number of times connections to the target data warehouse will be retried.
gg.eventhandler.snowflake.connectionRetryIntervalSeconds	Optional	Integer Value	30	Specifies the delay in seconds between connection retry attempts.

Table 10-43 (Cont.) Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .snowflake.deleteInsert	Optional	true or false	false	If set to true, Replicat will merge records using SQL DELETE+INSERT statements instead of SQL MERGE statement.

❶ Note
Applicable only if gg.ccompressseed.uodate is

Table 10-43 (Cont.) Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler.snowflake.detectMissingBaseRow	Optional	true or false	false	Diagnostic parameter to find UPDATE operations without base row. If set to true, Replicat will ABEND if there are UPDATE operations without base row. These rows will be collected into another table that can be investigated.
gg.eventhandler.snowflake.createTable	Optional	true or false	true	If the value is set to true, then target tables are automatically created if missing.
gg.eventhandler.snowflake.tableType	Optional	native or iceberg.	native	Indicates Snowflake table format for automatic table creation. Options are native (default): automatically create native Snowflake tables, and iceberg: automatically create Iceberg tables.

e
t
t
o
f
a
l
s
e
.

Table 10-43 (Cont.) Snowflake Event Handler Configuration

Properties	Required/Optional	Legal Values	Default	Explanation
gg.eventhandler .snowflake.icebergExternalVolume	Optional	External volume name	None	Required when the property <code>tableType</code> is set to <code>iceberg</code> . An external volume name to connect Snowflake to your external cloud storage for Iceberg tables.
gg.eventhandler .snowflake.icebergBaseLocation	Optional	Base location path	Empty ("")	Applicable when the property <code>tableType</code> is set to <code>iceberg</code> . Base location path specifies a relative path from the Iceberg table's external volume location, can be empty. Oracle GoldenGate will append the qualified table name to the base location path to create the Iceberg table.

10.2.37.1.3.2 Snowflake Storage Integration

When you use an external staging location, ensure to setup Snowflake storage integration to grant Snowflake database read permission to the files located in the cloud object store.

If the configuration property `gg.stage` is not set, then the storage integration is not required, and Oracle GoldenGate will default to internal stage.

- **Azure Data Lake Storage (ADLS) Gen2 Storage Integration:** For more information about creating the storage integration for Azure, see [Snowflake documentation to create the storage integration for Azure](#).

Example:

```
-- AS ACCOUNTADMIN
create storage integration azure_int
type = external_stage
storage_provider = azure
enabled = true
azure_tenant_id = '<azure tenant id>'
storage_allowed_locations = ('azure://<azure-account-name>.blob.core.windows.net/
<azure-container>/' );

desc storage integration azure_int;
-- Read AZURE_CONSENT_URL and accept the terms and conditions specified in the link.
-- Read AZURE_MULTI_TENANT_APP_NAME to get the Snowflake app name to be granted Blob
Read permission.
```

```
grant create stage on schema <schema name> to role <role name>;
grant usage on integration azure_int to role <role name>;
```

- **Google Cloud Storage (GCS) Storage Integration:** For more information about creating the storage integration for GCS, see [Snowflake Documentation](#).

Example:

```
create storage integration gcs_int
type = external_stage
storage_provider = gcs
enabled = true
storage_allowed_locations = ('gcs://<gcs-bucket-name>/' );

desc storage integration gcs_int;
-- Read the column STORAGE_GCP_SERVICE_ACCOUNT to get the GCP Service Account email
for Snowflake.
-- Create a GCP role with storage read permission and assign the role to the
Snowflake Service account.
```

```
grant create stage on schema <schema name> to role <role name>;
grant usage on integration gcs_int to role <role name>;
```

- **AWS S3 Storage Integration:** For more information about creating the storage integration for S3, see [Snowflake Documentation](#).

Note

When you use S3 as the external stage, you don't need to create storage integration if you already have access to the following AWS credentials: AWS Access Key Id and Secret key. You can set AWS credentials in the `jvm.bootoptions` property.

- The storage integration name must start with an alphabetic character and cannot contain spaces or special characters unless the entire identifier string is enclosed in double quotes for example, `My object`. Identifiers enclosed in double quotes are also case-sensitive.

10.2.37.1.3.3 Classpath Configuration

Snowflake Event Handler uses the Snowflake JDBC driver. Ensure that the classpath includes the path to the JDBC driver. If an external stage is used, then you need to also include the respective object store Event Handler's dependencies in the classpath.

- [Dependencies](#)

10.2.37.1.3.3.1 Dependencies

Snowflake JDBC driver: You can use the Dependency Downloader tool to download the JDBC driver by running the following script: `<OGGDIR>/DependencyDownloader/snowflake.sh`.

See Dependency Downloader for more information.

Alternatively, you can also download the JDBC driver from Maven central using the following [Dependency Downloader](#) co-ordinates:

```
<dependency>
  <groupId>net.snowflake</groupId>
  <artifactId>snowflake-jdbc</artifactId>
```

```
<version>3.13.19</version>  
</dependency>
```

- If staging location is set to S3, then the classpath should include the S3 Event handler dependencies. See [S3 Handler Configuration](#).
- If staging location is set to HDFS, then the classpath should include the HDFS Event handler dependencies. See [HDFS Event Handler Configuration](#).
- If staging location is set to Google Cloud Storage (GCS), then the classpath should include the GCS Event handler dependencies. See [Google Cloud Storage Event Handler Configuration](#).

Edit the `gg.classpath` configuration parameter to include the path to the object store Event Handler dependencies (if external stage is in use) and the Snowflake JDBC driver.

10.2.37.1.3.4 Proxy Configuration

When the Replicat process runs behind a proxy server, you can use the `jvm.bootoptions` property for proxy server configuration.

Example:

```
jvm.bootoptions=-Dhttp.useProxy=true -Dhttps.proxyHost=<some-proxy-address.com>  
-Dhttps.proxyPort=80 -Dhttp.proxyHost=<some-proxy-address.com> -Dhttp.proxyPort=80
```

10.2.37.1.3.5 INSERTALLRECORDS Support

Stage and merge targets supports `INSERTALLRECORDS` parameter.

See [INSERTALLRECORDS](#) in *Reference for Oracle GoldenGate*. Set the `INSERTALLRECORDS` parameter in the Replicat parameter file (`.prm`). Set the `INSERTALLRECORDS` parameter in the Replicat parameter file (`.prm`)

Setting this property directs the Replicat process to use bulk insert operations to load operation data into the target table. You can tune the batch size of bulk inserts using the File Writer property `gg.handler.snowflake.maxFileSize`. The default value is set to 1GB. The frequency of bulk inserts can be tuned using the File writer property `gg.handler.snowflake.fileRollInterval`, the default value is set to 3m (three minutes).

① Note

- When using the Snowflake internal stage, the staging files can be compressed by setting `gg.eventhandler.snowflake.putSQLAutoCompress` to `true`.
- Consider using the Snowflake Streaming Handler for INSERT-only workloads. See [Snowflake Streaming Handler](#).

10.2.37.1.3.6 Snowflake Key Pair Authentication

Snowflake supports key pair authentication as an alternative to basic authentication using username and password.

The path to the private key file must be set in the JDBC connection URL using the property: `private_key_file`.

If the private key file is encrypted, then the connection URL should also include the property: `private_key_file_pwd`.

Additionally, the connection URL should also include the Snowflake user that is assigned the respective public key by setting the property user.

Example JDBC connection URL:

```
jdbc:snowflake://<account_name>.snowflakecomputing.com/?warehouse=<warehouse-name>
&db=<database-name>&private_key_file=/path/to/private/key/rsa_key.p8
&private_key_file_pwd=<private-key-password>&user=<db-user>
```

When using key pair authentication, ensure that the Snowflake event handler parameters Username and Password are not set.

① Note

Oracle recommends that you upgrade Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) to version 21.10.0.0.0 or later. If you cannot upgrade to version 21.10.0.0.0 or later, update the JDBC URL by replacing \ characters with /.

10.2.37.1.3.7 Mapping Source JSON/XML to Snowflake VARIANT

The JSON and XML source column types in the Oracle GoldenGate trail gets automatically detected and mapped into Snowflake VARIANT.

You can inspect the metadata in the Oracle GoldenGate trail file for JSON and XML types using logdump.

Example: logdump output showing JSON and XML types:

```
022/01/06 01:38:54.717.464 Metadata          Len 679 RBA 6032
Table Name: CDB1_PDB1.TKGGU1.JSON_TAB1
*
  1)Name          2)Data Type          3)External Length  4)Fetch Offset
  5)Scale          6)Level
  7)Null           8)Bump if Odd          9)Internal Length 10)Binary Length  11)Table
Length 12)Most Sig DT
13)Least Sig DT 14)High Precision 15)Low Precision 16)Elementary Item
17)Occurs        18)Key Column
19)Sub DataType 20)Native DataType 21)Character Set 22)Character Length 23)LOB
Type            24)Partial Type
25)Remarks
*
TDR version: 11
Definition for table CDB1_PDB1.TKGGU1.JSON_TAB1
Record Length: 81624
Columns: 7
ID
50      50 0 0 0 0 1    0 1  2  2      -1      64    50      0 0 0 0 0    50
COL
8200    0 0 0 0 0 1    0 0  0 119    0      64    4000    56 0 0 1 0    4000
COL2
8200    0 0 0 0 0 1    0 0  0 119    0      64    4000    4062 0 0 1 0    4000
COL3
4000    0 0 0 0 0 1    0 0 10 112    -1     64    4000    8068 0 0 1 0    4000
SYS_NC00005$
4000    0 0 0 0 0 1    0 0  4 113    -1     64    8000    12074 0 0 1 0    4000
SYS_IME_OSON_CF27CFDF1CEB4FA2BF85A3D6239A433C
32767   0 0 0 0 0 1    0 0  4 23     -1     64    65534    16080 0 0 1 0    32767
SYS_IME_OSON_CEE1B31BB4494F6ABF31AC002BEBE941
32767   0 0 0 0 0 1    0 0  4 23     -1     64    65534    48852 0 0 1 0    32767
End of definition
```

In this example, COL and COL2 are JSON columns and COL3 is an XML column.

Additionally, mapping to Snowflake VARIANT is supported only if the source columns are stored as text.

10.2.37.1.3.8 Operation Aggregation

Operation aggregation is the process of aggregating (merging/compressing) multiple operations on the same row into a single output operation based on a threshold.

- [In-Memory Operation Aggregation](#)
- [Operation Aggregation Using SQL](#)

10.2.37.1.3.8.1 In-Memory Operation Aggregation

- Operation records are aggregated in-memory by default.
- The `gg.aggregate.operations.flush.interval` property has been deprecated and is no longer supported. If `gg.aggregate.operations.flush.interval` is used in GG for DAA 26ai, then replicat will run; but add a warning to log file about the property being deprecated and not supported.

To control the time window for aggregation, use `gg.handler.snowflake.fileRollInterval` property. By default, it is set to 3 minutes. Longer intervals will increase latency, and may increase memory usage. Shorter intervals will increase overhead in Oracle GoldenGate and the target database.

- Operation aggregation in-memory requires additional JVM memory configuration.

10.2.37.1.3.8.2 Operation Aggregation Using SQL

- To use SQL aggregation, it is mandatory that the trail files contain uncompressed UPDATE operation records, which means that the UPDATE operations contain full image of the row being updated.
- Operation aggregation using SQL can provide better throughput if the trails files contains uncompressed update records.
- Replicat can aggregate operations using SQL statements by setting the `gg.aggregate.operations.using.sql=true`.
- Operation aggregation using SQL does not require additional JVM memory configuration.

10.2.37.1.3.9 Compressed Update Handling

A compressed update record contains values for the key columns and the modified columns.

An uncompressed update record contains values for all the columns.

Oracle GoldenGate trails may contain compressed or uncompressed update records. The default extract configuration writes compressed updates to the trails. The parameter `gg.compressed.update` can be set to true or false to indicate compressed or uncompressed update records.

- [MERGE Statement with Uncompressed Updates](#)

10.2.37.1.3.9.1 MERGE Statement with Uncompressed Updates

In some use cases, if the trail contains uncompressed update records, then the MERGE SQL statement can be optimized for better performance by setting `gg.compressed.update=false`.

10.2.37.1.3.10 End-to-End Configuration

The following is an end-end configuration example which uses auto-configuration.

Location of the sample properties file: <OGGDIR>/AdapterExamples/big-data/snowflake/

- sf.props: Configuration using internal stage
- sf-s3.props: Configuration using S3 stage.
- sf-az.props: Configuration using ADLS Gen2 stage.
- sf-gcs.props: Configuration using GCS stage.

Note: Recommended to only edit the configuration marked as TODO

```
gg.target=snowflake
```

```
#The Snowflake Event Handler
```

```
#TODO: Edit JDBC ConnectionUrl
```

```
gg.eventhandler.snowflake.connectionURL=jdbc:snowflake://
```

```
<account_name>.snowflakecomputing.com/?warehouse=<warehouse-name>&db=<database-name>
```

```
#TODO: Edit JDBC user name
```

```
gg.eventhandler.snowflake.UserName=<db user name>
```

```
#TODO: Edit JDBC password
```

```
gg.eventhandler.snowflake.Password=<db password>
```

```
# Configuration to load GoldenGate trail operation records into Snowflake using  
Snowflake internal stage.
```

```
#TODO:Set the classpath to include Snowflake JDBC driver.
```

```
gg.classpath=$THIRD_PARTY_DIR/snowflake/*
```

```
#TODO:Provide sufficient memory.
```

```
jvm.bootoptions=-Xmx8g -Xms8g
```

```
# Configuration to load GoldenGate trail operation records into Snowflake using S3 stage.
```

```
#gg.stage=s3
```

```
#The S3 Event Handler
```

```
#TODO: Edit the AWS region
```

```
#gg.eventhandler.s3.region=<aws region>
```

```
#TODO: Edit the AWS S3 bucket
```

```
#gg.eventhandler.s3.bucketMappingTemplate=<s3 bucket>
```

```
#TODO:Set the classpath to include AWS Java SDK and Snowflake JDBC driver.
```

```
#gg.classpath=$THIRD_PARTY_DIR/s3/*:$THIRD_PARTY_DIR/snowflake/*
```

```
#TODO:Set the AWS access key and secret key. Provide sufficient memory.
```

```
#jvm.bootoptions=-Daws.accessKeyId=<AWS access key> -Daws.secretKey=<AWS secret key> -  
Xmx8g -Xms8g
```

```
# Configuration to load GoldenGate trail operation records into Snowflake using ADLS  
Gen2 stage.
```

```
#gg.stage=abs
```

```
# Azure Blob Event handler.
```

```
#gg.eventhandler.abs.bucketMappingTemplate=<azure_adls_gen2_container_name>
```

```
#gg.eventhandler.abs.accountName=<azure_storage_account_name>
```

```
#gg.eventhandler.abs.accountKey=<azure_storage_account_key>
```

```
#TODO: Edit snowflake storage integration to access Azure Blob Storage.
```

```
#gg.eventhandler.snowflake.storageIntegration=<azure_int>
```

```
#TODO: Edit the classpath to include HDFS Event Handler dependencies and Snowflake JDBC  
driver.
```

```
#gg.classpath=$THIRD_PARTY_DIR/abs/*:$THIRD_PARTY_DIR/snowflake/*
```

```
#TODO: Provide sufficient memory.
```

```
#jvm.bootoptions=-Xmx8g -Xms8g
```



```
# Configuration to load GoldenGate trail operation records into Snowflake using GCS
stage.
#gg.stage=gcs
## The GCS Event handler
#TODO: Edit the GCS bucket name
#gg.eventhandler.gcs.bucketMappingTemplate=<gcs bucket>
#TODO: Edit the GCS credentialsFile
#gg.eventhandler.gcs.credentialsFile=<oggbd-project-credentials.json>
#TODO: Edit snowflake storage integration to access GCS.
#gg.eventhandler.snowflake.storageIntegration=<gcs_int>
#TODO: Edit the classpath to include GCS Java SDK and Snowflake JDBC driver.
#gg.classpath=$THIRD_PARTY_DIR/gcs/*:$THIRD_PARTY_DIR/snowflake/*
#TODO: Provide sufficient memory.
#jvm.bootoptions=-Xmx8g -Xms8g
```

10.2.37.1.3.11 Table Mapping

If the MAP statement does not specify a target database, then the database set in the JDBC connection URL will be used. The handler will log the default database during initialization.

Example log message:

Connection catalog is set to [DB_1].

- [Mapping Table](#)

10.2.37.1.3.11.1 Mapping Table

Table 10-44 Snowflake Mapping Table

MAP statement in the Replicat parameter file	Snowflake Database	Snowflake Schema	Snowflake Table
MAP SCHEMA_1.TABLE_1, TARGET "schema_1"."table_1" ";	Default database	schema_1	table_1
MAP DB_1.SCHEMA_1.TABLE _1, TARGET "db_1"."schema_1"." table_1"	db_1	schema_1	table_1

10.2.37.1.4 Troubleshooting and Diagnostics

- **Connectivity issues to Snowflake:**
 - Validate JDBC connection URL, username, and password.
 - Check HTTP(S) proxy configuration if running Replicat process behind a proxy.
- **DDL not applied on the target table:** Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) does not support DDL replication.
- **SQL Errors:** In case there are any errors while executing any SQL, the SQL statements along with the bind parameter values are logged into the GG for DAA handler log file.

- **Co-existence of the components:** When using an external stage location (S3, ADLS Gen 2 or GCS), the location/region of the machine where the Replicat process is running and the object store's region have an impact on the overall throughput of the apply process. For the best possible throughput, the components need to be located ideally in the same region or as close as possible.
- **Replicat ABEND due to partial LOB records in the trail file:** GG for DAA does not support replication of partial LOB data. The trail file needs to be regenerated by Oracle Integrated capture using `TRANLOGOPTIONS FETCHPARTIALLOB` option in the Extract parameter file.
- When replicating to more than ten target tables, the parameter `maxConnections` can be increased to a higher value which can improve throughput.

Note

When tuning this, increasing the parameter value would create more JDBC connections on the Snowflake data warehouse. You can consult your Snowflake Database administrators so that the data warehouse health is not compromised.

- The Snowflake JDBC driver uses the standard Java log utility. The log levels of the JDBC driver can be set using the JDBC connection parameter `tracing`. The tracing level can be set in the Snowflake Event handler property `gg.eventhandler.snowflake.connectionURL`. The following is an example of editing this property:

```
jdbc:snowflake://<account_name>.snowflakecomputing.com/?
warehouse=<warehouse-name>&db=<database-name>&tracing=SEVERE
```

For more information, see <https://docs.snowflake.com/en/user-guide/jdbc-parameters.html#tracing>.

- **Exception: `net.snowflake.client.jdbc.SnowflakeReauthenticationRequest: Authentication token has expired. The user must authenticate again.`**
This error occurs when are extended periods of inactivity. To resolve this, you can set the JDBC parameter `CLIENT_SESSION_KEEP_ALIVE` to force the database user to login after a period of inactivity in the session. For example, `jdbc:snowflake://<account_name>.snowflakecomputing.com/?warehouse=<warehouse-name>&db=<database-name>&CLIENT_SESSION_KEEP_ALIVE=true`
- **Replicat stops with an out of memory error:** Decrease the `gg.aggregate.operations.flush.interval` value if you are not using its default value (30000).
- **Performance issue while replicating Large Object (LOB) column values:** LOB processing can lead to slowness. For every LOB column that exceeds the inline LOB threshold, an `UPDATE SQL` is executed. Look for the following message to tune throughput during LOB processing: The current operation at position [`<seqno>/<rba>`] for table [`<tablename>`] contains a LOB column [`<column name>`] of length [`<N>`] bytes that exceeds the threshold of maximum inline LOB size [`<N>`]. Operation Aggregator will flush merged operations, which can degrade performance. The maximum inline LOB size in bytes can be tuned using the configuration `gg.maxInlineLobSize`. Check the trail files that contain LOB data and get a maximum size of BLOB/CLOB columns. Alternatively, check the source table definitions to determine the maximum size of LOB data. The default inline LOB size is set to 16000 bytes, which can be increased to a higher value so that all LOB column updates are processed in batches. The configuration property is `gg.maxInlineLobSize``. For example: In

`gg.maxInlineLobSize=24000000` --> , all LOBs up to 24 MB are processed inline. You need to reposition the Replicat, purge the state files, data directory, and start over, so that bigger staging files are generated.

- Error message: No database is set in the current session. Please set a database in the JDBC connection url [gg.eventhandler.snowflake.connectionURL] using the option 'db=<database name>'.**
Resolution: Set the database name in the configuration property `gg.eventhandler.snowflake.connectionURL`.
- Warning message: No role is set in the current session. Please set a custom role name in the JDBC connection url [gg.eventhandler.snowflake.connectionURL] using the option 'role=<role name>' if the warehouse [{}] requires a custom role to access it.**
Resolution: In some cases a custom role is required to access the Snowflake warehouse, set the role in the configuration property `gg.eventhandler.snowflake.connectionURL`.
- Error message: No active warehouse selected in the current session. Please set the warehouse name (and custom role name if required to access the respective warehouse) in the JDBC connection url [gg.eventhandler.snowflake.connectionURL] using the options 'warehouse=<warehouse name>' and 'role=<role name>'.**
Resolution: Set the warehouse and role in the configuration property `gg.eventhandler.snowflake.connectionURL`.
- Error message: `ERROR 2024-06-07 05:52:23.000344 [main] - JDBCMDP-00034 Current attempt to connect failed with error: [Private key provided is invalid or not supported: ./rsa\_key\_sanav2.p8: PBE parameter parsing error: expecting the object identifier for AES cipher]`**
Resolution: This is a recent issue in the Snowflake JDBC driver. The workaround is to upgrade to the Snowflake JDBC driver version 3.16.1, and add the Java system property - `Dnet.snowflake.jdbc.enableBouncyCastle=true` to the `jvm.bootoptions` parameter in the Replicat properties file.
- Error message: org.apache.spark.SparkException: [FAILED_READ_FILE.FILE_NOT_EXIST] Error while reading file:** In cases of high load on the cluster it is observed that the refresh calls to refresh the contents of the external table are not reliable and hence leading to the above error resulting in a replicat abend. While Databricks fixes the JDBC driver to handle the refresh calls the alternate solution is to configure `gg.eventhandler.databricks.externalTableRefreshMethod=ALTER`.

10.2.37.1.5 Restrictions on Database Object Names

- Catalog and Schema Names:** Cannot contain space, period (.), asterisk (*), double quote ("), backtick (` ` `), question mark (?), backslash (\), tilde (~), open parenthesis ((), close parenthesis ()), pipe (|), semicolon (;), single quote ('), less than (<), greater than (>), comma (,), or control characters (ASCII 0-31).
- Table Names:** Cannot contain period (.), asterisk (*), question mark (?), double quote ("), backslash (\), or control characters (ASCII 0-31).
- Specific limitation for GGDA 21c and lower releases:** Supports only names that comply with the Avro specification for identifier names: must start with `[A-Za-z_]``, followed by `[A-Za-z0-9_]``

10.2.37.2 Snowflake Streaming Handler

- [Overview](#)

- [Detailed Functionality](#)
- [Configuration](#)
- [Classpath Configuration](#)
- [Proxy Configuration](#)
- [Sample Configuration](#)
- [Snowflake Streaming Handler Key Pair Authentication](#)
- [Troubleshooting and Diagnostics](#)

10.2.37.2.1 Overview

The Snowflake Streaming Handler replicates data into Snowflake using the Snowpipe Streaming API. This handler supports INSERT-only workloads using the Snowpipe Streaming API, which can result in lower load latencies at a lower cost for loading data into Snowflake.

Note

If your workload includes updates and deletes, consider using the [Snowflake Stage and Merge Handler](#).

For more information, see [Snowpipe Streaming](#) documentation.

10.2.37.2.2 Detailed Functionality

The change data from the Oracle GoldenGate trails is appended/streamed to a Snowflake target table using the Snowpipe Streaming API. The Streaming API provides low-latency loading of rows directly into the target table and also eliminates the need for a staging area.

Note

The Snowflake Streaming Handler supports INSERT-only workloads. You can leverage INSERTALLRECORDS to convert the update and delete statements if needed.

- [Database User Privileges](#)
- [Prerequisites](#)
- [Staging using Snowpipe Streaming API](#)

10.2.37.2.2.1 Database User Privileges

The database user used for replicating into Snowflake has to be granted the following privileges:

- INSERT on the target tables.
- Optionally, CREATE TABLE if setting `gg.handler.snow.createTable` to true.

10.2.37.2.2.2 Prerequisites

- Oracle GoldenGate trails must be configured to generate INSERT operations only. If update and delete operations are converted to inserts by using parameters like INSERTUPDATE, INSERTDELETE, or INSERTALLRECORDS then those inserts are also supported by handler.

10.2.37.2.2.3 Staging using Snowpipe Streaming API

The Snowpipe Streaming API allows for a low-latency ingest into the target table.

10.2.37.2.3 Configuration

The configuration of the Snowflake replication properties is stored in the Replicat properties file.

① Note

Ensure to specify the path to the properties file in the parameter file only when using Coordinated Replicat. Add the following line to the parameter file:

```
TARGETDB LIBFILE libggjava.so SET property=<parameter file directory>/
<properties file name>
```

The following are configuration properties available for the Snowflake Streaming handler, the required ones must be changed to match your Snowflake configuration.

Table 10-45 Snowflake Streaming Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handlerlist	Required	String value. For example, snow.	None	Choose the name snow for the handler.
gg.handler.<name>.type	Required	snowflakestreaming	None	Type of handler to use.
gg.handler.<name>.account	Required	String value.	None	Snowflake account name.
gg.handler.<name>.user	Required	String value.	None	Snowflake data warehouse user.
gg.handler.<name>.role	Optional	String value.	ACCOUNTADMIN	Snowflake data warehouse role.
gg.handler.<name>.warehouse	Required	String value.	None	Snowflake data warehouse name.
gg.handler.<name>.database	Required	String value.	None	Snowflake default database name used during connection.
gg.handler.<name>.privateKeyFile	Required	String value.	None	Specifies the fully qualified path to the private key file for the user. This is used for key-pair authentication.

Table 10-45 (Cont.) Snowflake Streaming Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.<name>.privateKeyFilePassword</code>	Optional	String value.	None	Specifies the password for the private key file in case the private key file is encrypted.
<code>gg.handler.<name>.createTable</code>	Optional	true or false	true	Set to true to automatically create the target table if it does not exist.
<code>gg.handler.<name>.flushTimeout</code>	Optional	Numeric value	30 seconds	Set the flush timeout for streaming operations to commit to target. The value should be in seconds.
<code>gg.handler.<name>.tableType</code>	Optional	native or iceberg.	native	Indicates Snowflake table format for automatic table creation. Options are native (default): automatically create native Snowflake tables, and iceberg: automatically create Iceberg tables.
<code>gg.handler.<name>.icebergExternalVolume</code>	Optional	External volume name	None	Required when the property <code>tableType</code> is set to iceberg. An external volume name to connect Snowflake to your external cloud storage for Iceberg tables.

Table 10-45 (Cont.) Snowflake Streaming Handler Configuration

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.<name>.icebergBaseLocation</code>	Optional	Base location path	Empty ("")	Applicable when the property <code>tableType</code> is set to <code>iceberg</code> . Base location path specifies a relative path from the Iceberg table's external volume location, can be empty. Oracle GoldenGate appends the qualified table name to the base location path to create the Iceberg table.
<code>gg.handler.<name>.catalogIntegrationName</code>	Optional	Catalog integration name	Empty ("")	Applicable when the property <code>tableType</code> is set to <code>iceberg</code> . Catalog integration name of the external catalog. If this is set, then the Snowflake internal Iceberg catalog gets synchronized with the external catalog. Any other SQL engine could use the external catalog to access the Iceberg tables.

10.2.37.2.4 Classpath Configuration

Snowflake Streaming Handler uses the Snowflake Ingest Java SDK. Ensure that the `gg.classpath` configuration parameter includes the path to the Ingest SDK.

- [Dependencies](#)
- [Maven Co-ordinates](#)

10.2.37.2.4.1 Dependencies

You can download the Dependency Downloader tool to download the dependencies by running the following script:

```
<OGGDIR>/DependencyDownloader/snowflake_streaming.sh.
```

For more information about Dependency Downloader, see [Dependency Downloader](#).

10.2.37.2.4.2 Maven Co-ordinates

Snowflake Ingest SDK:

```
<dependency>
  <groupId>net.snowflake</groupId>
  <artifactId>snowflake-ingest-sdk</artifactId>
  <version>2.1.0</version>
  <scope>provided</scope>
</dependency>
```

10.2.37.2.5 Proxy Configuration

When the Replicat process runs behind a proxy server, you can use the `jvm.bootoptions` property to set the proxy server configuration.

Example:

```
jvm.bootoptions=
-Dhttp.useProxy=true
-Dhttp.proxyHost=<some-proxy-address.com>
-Dhttp.proxy.port=<some-port-number>
```

10.2.37.2.6 Sample Configuration

The sample properties file can also be found in the directory `<OGGDIR>/AdapterExamples/big-data/snowflake_streaming/`.

Note: Recommended to only edit the configuration marked as TODO

```
gg.handlerlist=snow
gg.handler.snow.type=snowflakestreaming
#TODO: Edit database user.
gg.handler.snow.user=<db-user>
#TODO: Edit account name.
gg.handler.snow.account=<account-name>
#TODO: Edit role name.
gg.handler.snow.role=<role-name>
#TODO: Edit warehouse name.
gg.handler.snow.warehouse=<warehouse-name>
#TODO: Edit default database name.
gg.handler.snow.database=<default-db-name>
#TODO: Edit path to the private key file.
gg.handler.snow.privateKeyFile=/path/to/private/key/file/rsa_key.p8
#TODO: Edit password for the private key file.
gg.handler.snow.privateKeyFilePassword=<some-password>
#TODO: Set the classpath to include Snowflake ingest SDK and the Snowflake JDBC driver.
gg.classpath=.snowflake-ingest-sdk-2.1.1.jar
```

10.2.37.2.7 Snowflake Streaming Handler Key Pair Authentication

Snowflake Streaming API requires using key pair authentication. The path to the private key file must be set using the property: `gg.handler.snow.privateKeyFile`.

If the private key file is encrypted, specify its password using the property:

```
gg.handler.snow.privateKeyFilePassword
```

Additionally, include the Snowflake user that is assigned to the respective public key by setting the property `gg.handler.snow.user`.

You can perform the following steps for creating keys for Snowflake Key Pair Authentication:

1. Generate the private key using 3DES:

```
openssl genrsa 2048 | openssl pkcs8 -topk8 -v2 des3 -inform PEM -out  
rsa_key.p8
```

2. Generate the public key by referencing the private key:

```
openssl rsa -in rsa_key.p8 -pubout -out rsa_key.pub
```

3. Associate the public key with the Snowflake user:

```
ALTER USER <user> SET RSA_PUBLIC_KEY='key_value'
```

4. Configure the Snowflake JDBC URL:

```
jdbc:snowflake://<account_name>.snowflakecomputing.com  
/?warehouse=<warehouse-name>&db=<database-name>&role=<role>&user=<user>  
&private_key_file=/path/to/key/file/  
rsa_key.p8&private_key_file_pwd=<password>
```

Oracle recommends to use the latest Snowflake JDBC driver version for key pair authentication. For more information, see <https://docs.snowflake.com/en/user-guide/key-pair-auth>

10.2.37.2.8 Troubleshooting and Diagnostics

- **Connectivity issues to Snowflake:**
 - Validate configuration parameters: account, user, role, warehouse, privateKeyFile, privateKeyFilePassword, and database.
 - Check HTTP(S) proxy configuration if running Replicat process behind a proxy.
- **DDL not applied on the target table:** GG for DAA does not support DDL replication.
- **SQL Errors:** In case there are any errors while executing any SQL, the SQL statements along with the bind parameter values are logged into the GG for DAA handler log file.

10.2.37.3 Supported Snowflake Date Datatypes

Oracle GoldenGate supports the replication of following Snowflake Date Datatypes:

- DATE
- TIME
- DATETIME (mapped as VARCHAR)
- TIMESTAMP_NTZ
- TIMESTAMP_LTZ (mapped as TIMESTAMP_NTZ)
- TIMESTAMP_TZ

10.2.37.4 Snowflake Iceberg Tables

Apache Iceberg tables for Snowflake combine the performance and query semantics of typical Snowflake tables with external cloud storage that you manage.

They are ideal for existing data lakes that you cannot, or choose not to, store in Snowflake. For more information, see <https://docs.snowflake.com/en/user-guide/tables-iceberg>.

Oracle GoldenGate can automatically create Snowflake Iceberg tables by setting the property `gg.eventhandler.snowflake.tableType` to `iceberg`.

The property `gg.eventhandler.snowflake.icebergExternalVolume` must be set to the external volume name to connect Snowflake to your external cloud storage for Iceberg tables. The property `gg.eventhandler.snowflake.icebergBaseLocation` must be set to the base location path for the Snowflake Iceberg table.

Oracle GoldenGate will append the qualified table name to the base location path to create the Iceberg table.

For example:

```
CREATE ICEBERG TABLE "PUBLIC"."ICEBERG_TCUSTMER" (  
    "CUST_CODE" VARCHAR,  
    "NAME" VARCHAR,  
    "CITY" VARCHAR,  
    "STATE" VARCHAR,  
    PRIMARY KEY ("CUST_CODE")  
) CATALOG = 'SNOWFLAKE'  
EXTERNAL_VOLUME = 'iceberg_external_volume'  
BASE_LOCATION = 'SCHEMA_1/ICEBERG_TCUSTMER'  
CATALOG_SYNC = 'polaris_catalog_integration';
```

- [External Volume](#)
- [External Iceberg Catalog Synchronization](#)
- [Co-existence of Snowflake Native and Iceberg Tables](#)
- [Limitations](#)

10.2.37.4.1 External Volume

An external volume is a named, account-level Snowflake object that you use to connect Snowflake to your external cloud storage for Iceberg tables.

You can create an external volume in Snowflake by following the steps here: <https://docs.snowflake.com/en/user-guide/tables-iceberg-configure-external-volume>.

10.2.37.4.2 External Iceberg Catalog Synchronization

To query a Snowflake-managed Apache Iceberg table using a third-party engine such as Apache Spark, you can sync the table with an external Iceberg catalog. Currently, Polaris is only external Iceberg catalog that is supported by Snowflake for synchronization.

This can be enabled by setting the property `gg.eventhandler.snowflake.catalogIntegrationName` to the catalog integration name pointing to the external catalog.

You can create a catalog integration for an external catalog by using the `CREATE CATALOG INTEGRATION` SQL statement.

See <https://docs.snowflake.com/en/user-guide/tables-iceberg-polaris-sync> for steps to setup catalog synchronization.

10.2.37.4.3 Co-existence of Snowflake Native and Iceberg Tables

The co-existence of Snowflake native and Iceberg tables can occur provided the target tables exist in the same Snowflake warehouse, a single replicat process can replicate to both Snowflake native and Iceberg tables.

10.2.37.4.4 Limitations

- A default Snowflake external volume can be created at the Snowflake account-level, database-level, or schema-level. This can be overridden by setting the property `gg.eventhandler.snowflake.icebergExternalVolume`. External volume cannot be set at the table-level. All the tables in the same replicat process will use one external volume.
- A single replicat process cannot automatically create both Snowflake Iceberg tables and Snowflake native tables.

10.2.38 Additional Details

- [HDFS Event Handler](#)
The HDFS Event Handler is used to load files generated by the File Writer Handler into HDFS.
- [Metacolumn Keywords](#)
- [Metadata Providers](#)
The Metadata Providers can replicate from a source to a target using a Replicat parameter file.
- [Pluggable Formatters](#)
The pluggable formatters are used to convert operations from the Oracle GoldenGate trail file into formatted messages that you can send to Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) targets using one of the GG for DAA handlers.
- [Stage and Merge Data Warehouse Replication](#)
Data warehouse targets typically support Massively Parallel Processing (MPP). The cost of a single Data Manipulation Language (DML) operation is comparable to the cost of execution of batch DMLs.
- [Template Keywords](#)
- [Velocity Dependencies](#)
Starting Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) release 21.1.0.0.0, the Velocity jar files have been removed from the packaging.

10.2.38.1 HDFS Event Handler

The HDFS Event Handler is used to load files generated by the File Writer Handler into HDFS.

This topic describes how to use the HDFS Event Handler. See [Flat Files](#).

- [Detailing the Functionality](#)

10.2.38.1.1 Detailing the Functionality

- [Configuring the Handler](#)
- [Configuring the HDFS Event Handler](#)

10.2.38.1.1.1 Configuring the Handler

The HDFS Event Handler can upload data files to HDFS. These additional configuration steps are required:

The HDFS Event Handler dependencies and considerations are the same as the HDFS Handler, see [HDFS Additional Considerations](#).

Ensure that `gg.classpath` includes the HDFS client libraries.

Ensure that the directory containing the `HDFS core-site.xml` file is in `gg.classpath`. This is so the `core-site.xml` file can be read at runtime and the connectivity information to HDFS can be resolved. For example:

```
gg.classpath={HDFSinstallDirectory}/etc/hadoop
```

If Kerberos authentication is enabled on the HDFS cluster, you have to configure the Kerberos principal and the location of the keytab file so that the password can be resolved at runtime:

```
gg.eventHandler.name.kerberosPrincipal=principal
gg.eventHandler.name.kerberosKeytabFile=pathToTheKeytabFile
```

10.2.38.1.1.2 Configuring the HDFS Event Handler

You configure the HDFS Handler operation using the properties file. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

To enable the selection of the HDFS Event Handler, you must first configure the handler type by specifying `gg.eventhandler.name.type=hdfs` and the other HDFS Event properties as follows:

Table 10-46 HDFS Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.eventhandler.name.type</code>	Required	<code>hdfs</code>	None	Selects the HDFS Event Handler for use.
<code>gg.eventhandler.name.pathMappingTemplate</code>	Required	A string with resolvable keywords and constants used to dynamically generate the path in HDFS to write data files.	None	Use keywords interlaced with constants to dynamically generate unique path names at runtime. Path names typically follow the format, <code>/ogg/data/\${groupName}/\${fullyQualifiedTableName}</code> . See Template Keywords .

Table 10-46 (Cont.) HDFS Event Handler Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.eventhandler.name.fileNameMappingTemplate	Optional	A string with resolvable keywords and constants used to dynamically generate the HDFS file name at runtime.	None	Use keywords interlaced with constants to dynamically generate unique file names at runtime. If not set, the upstream file name is used. See Template Keywords .
gg.eventhandler.name.finalizeAction	Optional	none delete	none	Indicates what the File Writer Handler should do at the finalize action. none Leave the data file in place (removing any active write suffix, see About the Active Write Suffix). delete Delete the data file (such as, if the data file has been converted to another format or loaded to a third party application).
gg.eventhandler.name.kerberosPrincipal	Optional	The Kerberos principal name.	None	Set to the Kerberos principal when HDFS Kerberos authentication is enabled.
gg.eventhandler.name.keberosKeytabFile	Optional	The path to the Keberos keytab file.	None	Set to the path to the Kerberos keytab file when HDFS Kerberos authentication is enabled.
gg.eventhandler.name.eventHandler	Optional	A unique string identifier cross referencing a child event handler.	No event handler configured.	A unique string identifier cross referencing an event handler. The event handler will be invoked on the file roll event. Event handlers can do thing file roll event actions like loading files to S3, converting to Parquet or ORC format, or loading files to HDFS.

10.2.38.2 Metacolumn Keywords

This appendix describes the metacolumn keywords.

The metacolumns functionality allows you to select the metadata fields that you want to see in the generated output messages. The format of the metacolumn syntax is:

`${keyword[fieldName].argument}`

The keyword is fixed based on the metacolumn syntax. Optionally, you can provide a field name between the square brackets. If a field name is not provided, then the default field name is used.

Keywords are separated by a comma. Following is an example configuration of metacolumns:

```
gg.handler.filewriter.format.metaColumnsTemplate=${objectname[table]},${
optype[op_type]},${timestamp[op_ts]},${currenttimestamp[current_ts]},${
position[pos]}
```

An argument may be required for a few metacolumn keywords. For example, it is required where specific token values are resolved or specific environmental variable values are resolved.

`${alltokens}`

All of the tokens for an operation delivered as a map where the token keys are the keys in the map and the token values are the map values.

`${token}`

The value of a specific Oracle GoldenGate token. The token key should follow token key should follow the token using the period (.) operator. For example:

```
${token.MYTOKEN}
```

`${sys}`

A system environmental variable. The variable name should follow sys using the period (.) operator.

`${sys.MYVAR}`

An Oracle GoldenGate environment variable. The variable name should follow env using the period (.) operator.

`${env}`

An Oracle GoldenGate environment variable. The variable name should follow env using the period (.) operator. For example:

```
${env.someVariable}
```

`${javaprop}`

A Java JVM variable. The variable name should follow javaprop using the period (.) operator. For example:

```
${javaprop.MYVAR}
```

`${optype}`

The operation type. This is generally I for inserts, U for updates, D for deletes, and T for truncates.

`${position}`

The record position. This is location of the record in the source trail file. It is a 20 character string. The first 10 characters is the trail file sequence number. The last 10 characters is the offset or rba of the record in the trail file.

`${timestamp}`

Record timestamp.

\${catalog}

Catalog name.

\${schema}

Schema name.

\${table}

Table name.

\${objectname}

The fully qualified table name.

\${csn}

Source Commit Sequence Number.

\${xid}

Source transaction ID.

\${currenttimestamp}

Current timestamp.

\${currenttimestampiso8601}

Current timestamp in ISO 8601 format.

\${opseqno}

Record sequence number within the transaction.

\${timestampmicro}

Record timestamp in microseconds after epoch.

\${currenttimestampmicro}

Current timestamp in microseconds after epoch.

\${txind}

This is the transactional indicator from the source trail file. The values of a transaction are B for the first operation, M for the middle operations, E for the last operation, or W for whole if there is only one operation. Filtering operations or the use of coordinated apply negate the usefulness of this field.

\${primarykeycolumns}

Use to inject a field with a list of the primary key column names.

\${primarykeys}

Use to inject a field with a list of the primary key column values with underscore (_) delimiter between primary key values.

Usage: \${primarykeys[fieldName]}

Example: \${primarykeys[JMSXGroupID]}

\${static}

Use to inject a field with a static value into the output. The value desired should be the argument. If the desired value is abc, then the syntax is \${static.abc} or \${static[FieldName].abc}.

\${seqno}

Used to inject a field containing the sequence number of the source trail file for the given operation.

\${rba}

Used to inject a field containing the rba (offset) of the operation in the source trail file for the given operation.

\${metadatachanged}

A boolean field which gets set to true on the first operation following a metadata change for the source table definition.

\${groupname}

A string field which the value is the group name of the replicat process. Group name is effectively the replicat process name as it is referred to in ggsci or the Oracle GoldenGate Microservices UI.

\${positionnumber}

The position rendered as a number.

\${seqnonumber}

The trail sequence number rendered as a number.

\${rbanumber}

The trail rba rendered as a number.

\${opseqno}

The operation sequence number rendered as a number.

\${primarykeysmmap}

Resolves a metacolumn where the value of the metacolumn is a map of the primary keys. The keys in the map are the primary key column names and the values are the primary key column values represented as string. For example:

```
{
  "primarykeysmmap": {
    "ORDER_DATE": "1994-09-30:15:33:00",
    "ORDER_ID": "144",
    "CUST_CODE": "WILL",
    "PRODUCT_CODE": "CAR"
  }
}
```

Alternatively, the metacolumn name can be changed so `${primarykeysmmap[pks]}` would generate the following:

```
{
  "pks": {
    "ORDER_DATE": "1994-09-30:15:33:00",
    "ORDER_ID": "144",
    "CUST_CODE": "WILL",
    "PRODUCT_CODE": "CAR"
  }
}
```

10.2.38.3 Metadata Providers

The Metadata Providers can replicate from a source to a target using a Replicat parameter file.

This chapter describes how to use the Metadata Providers.

- [About the Metadata Providers](#)
- [Avro Metadata Provider](#)
The Avro Metadata Provider is used to retrieve the table metadata from Avro Schema files. For every table mapped in Replicat using COLMAP, the metadata is retrieved from Avro Schema. Retrieved metadata is then used by Replicat for column mapping.
- [Cassandra Metadata Provider](#)
The Cassandra metadata provider is used to retrieve the table metadata from the Cassandra instance. The metadata is retrieved from Cassandra for every target table that is mapped in the replicat properties file using the COLMAP parameter. The keyspace and tables should already be created on the target for Cassandra MDP to fetch the metadata.
- [Java Database Connectivity Metadata Provider](#)
The Java Database Connectivity (JDBC) Metadata Provider is used to retrieve the table metadata from any target database that supports a JDBC connection and has a database schema. It is the preferred metadata provider for any target RDBMS database, although various other non-RDBMS targets also provide a JDBC driver.
- [Hive Metadata Provider](#)
The Hive Metadata Provider is used to retrieve the table metadata from a Hive metastore. The metadata is retrieved from Hive for every target table that is mapped in the Replicat properties file using the COLMAP parameter. The retrieved target metadata is used by Replicat for the column mapping functionality.
- [Google BigQuery Metadata Provider](#)
Google metadata provider uses the Google Query Job to retrieve the metadata schema information from the Google BigQuery Table. The Table should already be created on the target for BigQuery to fetch the metadata.

10.2.38.3.1 About the Metadata Providers

Metadata Providers work only if handlers are configured to run with a Replicat process.

The Replicat process maps source table to target table and source column to target column mapping using syntax in the Replicat configuration file. The source metadata definitions are included in the Oracle GoldenGate trail file (or by source definitions files in Oracle GoldenGate releases 12.2 and later). When the replication target is a database, the Replicat process obtains the target metadata definitions from the target database. However, this is a shortcoming when pushing data to Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) applications or during Java delivery in general. Typically, GG for DAA applications provide no target metadata, so Replicat mapping is not possible. The metadata providers exist to address this deficiency. You can use a metadata provider to define target metadata using either Avro or Hive, which enables Replicat mapping of source table to target table and source column to target column.

The use of the metadata provider is optional and is enabled if the `gg.mdp.type` property is specified in the Java Adapter Properties file. If the metadata included in the source Oracle GoldenGate trail file is acceptable for output, then do not use the metadata provider. Use a metadata provider should be used in the following cases:

- You need to map source table names into target table names that do not match.
- You need to map source column names into target column name that do not match.
- You need to include certain columns from the source trail file and omit other columns.

A limitation of Replicat mapping is that the mapping defined in the Replicat configuration file is static. Oracle GoldenGate provides functionality for DDL propagation when using an Oracle

database as the source. The proper handling of schema evolution can be problematic when the Metadata Provider and Replicat mapping are used. Consider your use cases for schema evolution and plan for how you want to update the Metadata Provider and the Replicat mapping syntax for required changes.

For every table mapped in Replicat using COLMAP, the metadata is retrieved from a configured metadata provider and retrieved metadata then be used by Replicat for column mapping.

Only the Hive and Avro Metadata Providers are supported and you must choose one or the other to use in your metadata provider implementation.

Scenarios - When to use a metadata provider

1. The following scenarios do *not* require a metadata provider to be configured:

A mapping in which the source schema named GG is mapped to the target schema named GGADP.*

A mapping in which the schema and table name whereby the schema GG.TCUSTMER is mapped to the table name GGADP.TCUSTMER_NEW

```
MAP GG.*, TARGET GGADP.*;
(OR)
MAP GG.TCUSTMER, TARGET GG_AD.P.TCUSTMER_NEW;
```

2. The following scenario requires a metadata provider to be configured:

A mapping in which the source column name does not match the target column name. For example, a source column of CUST_CODE mapped to a target column of CUST_CODE_NEW.

```
MAP GG.TCUSTMER, TARGET GG_AD.P.TCUSTMER_NEW, COLMAP(USEDEFAULTS,
CUST_CODE_NEW=CUST_CODE, CITY2=CITY);
```

10.2.38.3.2 Avro Metadata Provider

The Avro Metadata Provider is used to retrieve the table metadata from Avro Schema files. For every table mapped in Replicat using COLMAP, the metadata is retrieved from Avro Schema. Retrieved metadata is then used by Replicat for column mapping.

- [Detailed Functionality](#)
- [Runtime Prerequisites](#)
- [Classpath Configuration](#)
- [Avro Metadata Provider Configuration](#)
- [Review a Sample Configuration](#)
- [Metadata Change Events](#)
- [Limitations](#)
- [Troubleshooting](#)

10.2.38.3.2.1 Detailed Functionality

The Avro Metadata Provider uses Avro schema definition files to retrieve metadata. Avro schemas are defined using JSON. For each table mapped in the *process_name*.prm file, you must create a corresponding Avro schema definition file.

Avro Metadata Provider Schema Definition Syntax

```
{"namespace": "[$catalogname.]$schemaname",
"type": "record",
```

```

"name": "$tablename",
"fields": [
  {"name": "$col1", "type": "$datatype"},
  {"name": "$col2 ", "type": "$datatype ", "primary_key":true},
  {"name": "$col3", "type": "$datatype ", "primary_key":true},
  {"name": "$col4", "type": ["$datatype","null"]}
]
}

```

namespace - name of catalog/schema being mapped
 name - name of the table being mapped
 fields.name - array of column names
 fields.type - datatype of the column
 fields.primary_key - indicates the column is part of primary key.

Representing nullable and not nullable columns:

"type": "\$datatype" - indicates the column is not nullable, where "\$datatype" is the actual datatype.
 "type": ["\$datatype","null"] - indicates the column is nullable, where "\$datatype" is the actual datatype

The names of schema files that are accessed by the Avro Metadata Provider must be in the following format:

[*\$catalogname*.]*\$schemaname*.*\$tablename*.mdp.avsc

\$catalogname - name of the catalog if exists
\$schemaname - name of the schema
\$tablename - name of the table
 .mdp.avsc - constant, which should be appended always

Supported Avro Primitive Data Types

- boolean
- bytes
- double
- float
- int
- long
- string

See https://avro.apache.org/docs/1.7.5/spec.html#schema_primitive.

Supported Avro Logical Data Types

- decimal
- timestamp

Example Avro for decimal logical type

```

{"name": "DECIMALFIELD", "type":
{"type": "bytes", "logicalType": "decimal", "precision": 15, "scale": 5}}

```

Example of Timestamp logical type

```

{"name": "TIMESTAMPFIELD", "type":
{"type": "long", "logicalType": "timestamp-micros"}}

```

10.2.38.3.2.2 Runtime Prerequisites

Before you start the Replicat process, create Avro schema definitions for all tables mapped in Replicat's parameter file.

10.2.38.3.2.3 Classpath Configuration

The Avro Metadata Provider requires no additional classpath setting.

10.2.38.3.2.4 Avro Metadata Provider Configuration

Property	Required/ Optional	Legal Values	Default	Explanation
gg.mdp.type	Required	avro	-	Selects the Avro Metadata Provider
gg.mdp.schema FilePath	Required	Example: /home/user/ ggadp/avroschema/	-	The path to the Avro schema files directory
gg.mdp.charse t	Optional	Valid character set	UTF-8	Specifies the character set of the column with character data type. Used to convert the source data from the trail file to the correct target character set.
gg.mdp.nation alCharset	Optional	Valid character set	UTF-8	Specifies the character set of the column with character data type. Used to convert the source data from the trail file to the correct target character set. Example: Used to indicate character set of columns, such as NCHAR, NVARCHAR in an Oracle database.

10.2.38.3.2.5 Review a Sample Configuration

This is an example for configuring the Avro Metadata Provider. Consider a source that includes the following table:

```
TABLE GG.TCUSTMER {
  CUST_CODE VARCHAR(4) PRIMARY KEY,
  NAME VARCHAR(100),
  CITY VARCHAR(200),
  STATE VARCHAR(200)
}
```

This table maps the(CUST_CODE (GG.TCUSTMER) in the source to CUST_CODE2 (GG_AVRO.TCUSTMER_AVRO) on the target and the column CITY(GG.TCUSTMER) in source to CITY2 (GG_AVRO.TCUSTMER_AVRO) on the target. Therefore, the mapping in the *process_name*.prm file is:

```
MAP GG.TCUSTMER, TARGET GG_AVRO.TCUSTMER_AVRO, COLMAP(USEDEFAULTS, CUST_CODE2=CUST_CODE,
CITY2=CITY);
```

In this example the mapping definition is as follows:

- Source schema GG is mapped to target schema GG_AVRO.
- Source column CUST_CODE is mapped to target column CUST_CODE2.
- Source column CITY is mapped to target column CITY2.
- USEDEFAULTS specifies that rest of the columns names are same on both source and target (NAME and STATE columns).

This example uses the following Avro schema definition file:

File path: /home/ggadp/avromdpGG_AVRO.TCUSTMER_AVRO.mdp.avsc

```
{ "namespace": "GG_AVRO",
  "type": "record",
  "name": "TCUSTMER_AVRO",
  "fields": [
    { "name": "NAME", "type": "string" },
    { "name": "CUST_CODE2", "type": "string", "primary_key": true },
    { "name": "CITY2", "type": "string" },
    { "name": "STATE", "type": [ "string", "null" ] }
  ]
}
```

The configuration in the Java Adapter properties file includes the following:

```
gg.mdp.type = avro
gg.mdp.schemaFilePath = /home/ggadp/avromdp
```

The following sample output uses a delimited text formatter with a semi-colon as the delimiter:

```
I;GG_AVRO.TCUSTMER_AVRO;2013-06-02 22:14:36.000000;NAME;BG SOFTWARE
CO;CUST_CODE2;WILL;CITY2;SEATTLE;STATE;WA
```

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) includes a sample Replicat configuration file, a sample Java Adapter properties file, and sample Avro schemas at the following location:

GoldenGate_install_directory/AdapterExamples/big-data/metadata_provider/avro

10.2.38.3.2.6 Metadata Change Events

If the DDL changes in the source database tables, you may need to modify the Avro schema definitions and the mappings in the Replicat configuration file. You may also want to stop or suspend the Replicat process in the case of a metadata change event. You can stop the Replicat process by adding the following line to the Replicat configuration file (*process_name.prm*):

```
DDL INCLUDE ALL, EVENTACTIONS (ABORT)
```

Alternatively, you can suspend the Replicat process by adding the following line to the Replication configuration file:

```
DDL INCLUDE ALL, EVENTACTIONS (SUSPEND)
```

10.2.38.3.2.7 Limitations

Avro bytes data type cannot be used as primary key.

The source-to-target mapping that is defined in the Replicat configuration file is static. Oracle GoldenGate 12.2 and later support DDL propagation and source schema evolution for Oracle Databases as replication source. If you use DDL propagation and source schema evolution, you lose the ability to seamlessly handle changes to the source metadata.

10.2.38.3.2.8 Troubleshooting

This topic contains the information about how to troubleshoot the following issues:

- [Invalid Schema Files Location](#)
- [Invalid Schema File Name](#)
- [Invalid Namespace in Schema File](#)
- [Invalid Table Name in Schema File](#)

10.2.38.3.2.8.1 Invalid Schema Files Location

The Avro schema files directory specified in the `gg.mdp.schemaFilePath` configuration property must be a valid directory. If the path is not valid, you encounter following exception:

```
oracle.goldengate.util.ConfigException: Error initializing Avro metadata provider  
Specified schema location does not exist. {/path/to/schema/files/dir}
```

10.2.38.3.2.8.2 Invalid Schema File Name

For every table that is mapped in the `process_name.prm` file, you must create a corresponding Avro schema file in the directory that is specified in `gg.mdp.schemaFilePath`.

For example, consider the following scenario:

Mapping:

```
MAP GG.TCUSTMER, TARGET GG_AVRO.TCUSTMER_AVRO, COLMAP(USEDEFAULTS, cust_code2=cust_code,  
CITY2 = CITY);
```

Property:

```
gg.mdp.schemaFilePath=/home/usr/avro/
```

In this scenario, you must create a file called `GG_AVRO.TCUSTMER_AVRO.mdp.avsc` in the `/home/usr/avro/` directory.

If you do not create the `/home/usr/avro/GG_AVRO.TCUSTMER_AVRO.mdp.avsc` file, you encounter the following exception:

```
java.io.FileNotFoundException: /home/usr/avro/GG_AVRO.TCUSTMER_AVRO.mdp.avsc
```

10.2.38.3.2.8.3 Invalid Namespace in Schema File

The target schema name specified in Replicat mapping must be same as the namespace in the Avro schema definition file.

For example, consider the following scenario:

Mapping:

```
MAP GG.TCUSTMER, TARGET GG_AVRO.TCUSTMER_AVRO, COLMAP(USEDEFAULTS, cust_code2 =  
cust_code, CITY2 = CITY);
```

Avro Schema Definition:

```
{  
  "namespace": "GG_AVRO",  
  ..  
}
```

In this scenario, Replicat abends with following exception:

```
Unable to retrieve table matadata. Table : GG_AVRO.TCUSTMER_AVRO  
Mapped [catalogname.]schemaname (GG_AVRO) does not match with the schema namespace  
{schema namespace}
```

10.2.38.3.2.8.4 Invalid Table Name in Schema File

The target table name that is specified in Replicat mapping must be same as the name in the Avro schema definition file.

For example, consider the following scenario:

Mapping:

```
MAP GG.TCUSTMER, TARGET GG_AVRO.TCUSTMER_AVRO, COLMAP(USEDEFAULTS, cust_code2 =  
cust_code, CITY2 = CITY);
```

Avro Schema Definition:

```
{  
  "namespace": "GG_AVRO",  
  "name": "TCUSTMER_AVRO",  
  ..  
}
```

In this scenario, if the target table name specified in Replicat mapping does not match with the Avro schema name, then REPLICAT abends with following exception:

```
Unable to retrieve table matadata. Table : GG_AVRO.TCUSTMER_AVRO  
Mapped table name (TCUSTMER_AVRO) does not match with the schema table name {table name}
```

10.2.38.3.3 Cassandra Metadata Provider

The Cassandra metadata provider is used to retrieve the table metadata from the Cassandra instance. The metadata is retrieved from Cassandra for every target table that is mapped in the replicat properties file using the COLMAP parameter. The keyspace and tables should already be created on the target for Cassandra MDP to fetch the metadata.

The metadata retrieved from the Cassandra target includes primary and partition key definitions as well. All columns retrieved from Cassandra target are by default marked as nullable by the metadata provider.

- [Supported Cassandra Data Types](#)
- [Unsupported Cassandra Data Types](#)
- [Configuration](#)
- [Sample Configuration](#)
- [Limitations](#)

10.2.38.3.3.1 Supported Cassandra Data Types

- BLOB
- BOOLEAN
- DECIMAL
- DOUBLE
- FLOAT
- INET
- ASCII
- TEXT
- VARCHAR
- TINYINT
- SMALLINT
- INT
- BIGINT
- VARINT
- DURATION
- TIME
- TIMESTAMP
- DATE
- TIMEUUID

10.2.38.3.3.2 Unsupported Cassandra Data Types

- COUNTER
- MAP
- SET
- LIST
- UDT (user defined type)
- TUPLE
- CUSTOM_TYPE
- TIMESTAMP
- DATE
- TIMEUUID

10.2.38.3.3.3 Configuration

The Cassandra Driver location must be included in the class path of the handler using the `gg.classpath` property.

Table 10-47 Cassandra Metadata Provider Configuration

Properties	Required/ Optional	Legal values	Default	Explanation
gg.mdp.type	Required	cassandra	None	Entering cassandra at a command prompt activates the use of the Cassandra Metadata Provider.
gg.mdp.contactPoints	Required	A comma separated list of host names that the Cassandra Handler will connect to.	None	A comma-separated list of the Cassandra host machines for the driver to establish an initial connection to the Cassandra cluster. This configuration property does not need to include all the machines enlisted in the Cassandra cluster. By connecting to a single machine, the driver can learn about other machines in the Cassandra cluster and establish connections to those machines as required.
gg.mdp.port	Optional	Integer	9042	Set to configure the port number that the Cassandra MDP attempts to connect to Cassandra server instances. You can override the default in the Cassandra YAML files.
gg.mdp.datacenter	Optional	The datacenter name	None	Set the datacenter name. If the datacenter name does not match the configured name on the server, then it will not connect to the database.
gg.mdp.username	Optional	A legal username string.	None	A username for the connection to name. Required if Cassandra is configured to require credentials.

Table 10-47 (Cont.) Cassandra Metadata Provider Configuration

Properties	Required/ Optional	Legal values	Default	Explanation
gg.mdp.password	Optional	A legal password string.	None	A password for the connection to name. Required if Cassandra is configured to require credentials.
gg.handler.name .DatastaxJSSECo nfigPath	Optional	String	None	Set the path and file name of a properties file containing the Cassandra driver configuration. Use when the Cassandra driver configuration needs to be configured for non-default values and potentially SSL connectivity.

10.2.38.3.3.4 Sample Configuration

Sample Properties File Content

```
gg.mdp.type=cassandra
gg.mdp.datacenter=datacenter1
gg.mdp.contactPoints=localhost
```

Sample Parameter File Content

```
REPLICAT cassandra
MAP schema.tableName, TARGET keyspace.tableName;
```

10.2.38.3.3.5 Limitations

The Cassandra handler table auto-creation module does not work with the Cassandra Metadata provider as it expects the schema to be already created in the Cassandra instance. Complex Cassandra data types, such as LIST, MAP, and STRUCT are not yet supported.

10.2.38.3.4 Java Database Connectivity Metadata Provider

The Java Database Connectivity (JDBC) Metadata Provider is used to retrieve the table metadata from any target database that supports a JDBC connection and has a database schema. It is the preferred metadata provider for any target RDBMS database, although various other non-RDBMS targets also provide a JDBC driver.

- [JDBC Detailed Functionality](#)
- [Java Classpath](#)
- [JDBC Metadata Provider Configuration](#)
- [Review a Sample Configuration](#)

10.2.38.3.4.1 JDBC Detailed Functionality

The JDBC Metadata Provider uses the JDBC driver that is provided with your target database. The JDBC driver retrieves the metadata for every target table that is mapped in the Replicat properties file. Replicat processes use the retrieved target metadata to map columns.

You can enable this feature for JDBC Handler by configuring the `REPERROR` property in your Replicat parameter file. In addition, you need to define the error codes specific to your RDBMS JDBC target in the JDBC Handler properties file as follows:

Table 10-48 JDBC REPERROR Codes

Property	Value	Required
<code>gg.error.duplicateErrorCodes</code>	Comma-separated integer values of error codes that indicate duplicate errors	No
<code>gg.error.notFoundErrorCodes</code>	Comma-separated integer values of error codes that indicate Not Found errors	No
<code>gg.error.deadlockErrorCodes</code>	Comma-separated integer values of error codes that indicate deadlock errors	No

For example:

```
#ErrorCode
gg.error.duplicateErrorCodes=1062,1088,1092,1291,1330,1331,1332,1333
gg.error.notFoundErrorCodes=0
gg.error.deadlockErrorCodes=1213
```

To understand how the various JDBC types are mapped to database-specific SQL types, see <https://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/getstart/mapping.html#table1>.

10.2.38.3.4.2 Java Classpath

The JDBC Java Driver location must be included in the class path of the handler using the `gg.classpath` property.

For example, the configuration for a MySQL database might be:

```
gg.classpath= /path/to/jdbc/driver/jar/mysql-connector-java-5.1.39-bin.jar
```

10.2.38.3.4.3 JDBC Metadata Provider Configuration

The following are the configurable values for the JDBC Metadata Provider. These properties are located in the Java Adapter properties file (not in the Replicat properties file).

Table 10-49 JDBC Metadata Provider Properties

Properties	Required/Optional	Legal Values	Default	Explanation
<code>gg.mdp.type</code>	Required	<code>jdbc</code>	None	Entering <code>jdbc</code> at a command prompt activates the use of the JDBC Metadata Provider.

Table 10-49 (Cont.) JDBC Metadata Provider Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.mdp.Connect ionUrl	Required	<code>jdbc:subp rotocol:s ubname</code>	None	The target database JDBC URL.
gg.mdp.DriverC lassName	Required	Java class name of the JDBC driver	None	The fully qualified Java class name of the JDBC driver.
gg.mdp.userNam e	Optional	A legal username string.	None	The user name for the JDBC connection. Alternatively, you can provide the user name using the ConnectionURL property.
gg.mdp.passwor d	Optional	A legal password string	None	The password for the JDBC connection. Alternatively, you can provide the password using the ConnectionURL property.

10.2.38.3.4.4 Review a Sample Configuration

MySQL Driver Configuration

```
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:oracle:thin:@myhost:1521:orcl
gg.mdp.DriverClassName=oracle.jdbc.driver.OracleDriver
gg.mdp.UserName=username
gg.mdp.Password=password
```

Netezza Driver Configuration

```
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:netezza://hostname:port/databaseName
gg.mdp.DriverClassName=org.netezza.Driver
gg.mdp.UserName=username
gg.mdp.Password=password
```

Oracle OCI Driver configuration

```
ggg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:oracle:oci:@myhost:1521:orcl
gg.mdp.DriverClassName=oracle.jdbc.driver.OracleDriver
gg.mdp.UserName=username
gg.mdp.Password=password
```

Oracle Teradata Driver configuration

```
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:teradata://10.111.11.111/USER=username,PASSWORD=password
gg.mdp.DriverClassName=com.teradata.jdbc.TeraDriver
gg.mdp.UserName=username
gg.mdp.Password=password
```

Oracle Thin Driver Configuration

```
gg.mdp.type=jdbc
gg.mdp.ConnectionUrl=jdbc:mysql://localhost/databaseName?user=username&password=password
```

```
gg.mdp.DriverClassName=com.mysql.jdbc.Driver  
gg.mdp.UserName=username  
gg.mdp.Password=password
```

Redshift Driver Configuration

```
gg.mdp.type=jdbc  
gg.mdp.ConnectionUrl=jdbc:redshift://hostname:port/databaseName  
gg.mdp.DriverClassName=com.amazon.redshift.jdbc42.Driver  
gg.mdp.UserName=username  
gg.mdp.Password=password
```

10.2.38.3.5 Hive Metadata Provider

The Hive Metadata Provider is used to retrieve the table metadata from a Hive metastore. The metadata is retrieved from Hive for every target table that is mapped in the Replicat properties file using the COLMAP parameter. The retrieved target metadata is used by Replicat for the column mapping functionality.

- [Detailed Functionality](#)
- [Configuring Hive with a Remote Metastore Database](#)
- [Classpath Configuration](#)
- [Hive Metadata Provider Configuration Properties](#)
- [Review a Sample Configuration](#)
- [Security](#)
- [Metadata Change Event](#)
- [Limitations](#)
- [Additional Considerations](#)
- [Troubleshooting](#)

10.2.38.3.5.1 Detailed Functionality

The Hive Metadata Provider uses both Hive JDBC and HCatalog interfaces to retrieve metadata from the Hive metastore. For each table mapped in the *process_name.prm* file, a corresponding table is created in Hive.

The default Hive configuration starts an embedded, local metastore Derby database. Because, Apache Derby is designed to be an embedded database, it allows only a single connection. The limitation of the Derby Database means that it cannot function when working with the Hive Metadata Provider. To work around this limitation this, you must configure Hive with a remote metastore database. For more information about how to configure Hive with a remote metastore database, see <https://cwiki.apache.org/confluence/display/Hive/AdminManual+Metastore+Administration>.

Hive does not support Primary Key semantics, so the metadata retrieved from Hive metastore does not include a primary key definition. When you use the Hive Metadata Provider, use the Replicat KEYCOLS parameter to define primary keys.

KEYCOLS

Use the KEYCOLS parameter must be used to define primary keys in the target schema. The Oracle GoldenGate HBase Handler requires primary keys. Therefore, you must set primary keys in the target schema when you use Replicat mapping with HBase as the target.

The output of the Avro formatters includes an Array field to hold the primary column names. If you use Replicat mapping with the Avro formatters, consider using KEYCOLS to identify the primary key columns.

For example configurations of KEYCOLS, see [Review a Sample Configuration](#).

Supported Hive Data types

- BIGINT
- BINARY
- BOOLEAN
- CHAR
- DATE
- DECIMAL
- DOUBLE
- FLOAT
- INT
- SMALLINT
- STRING
- TIMESTAMP
- TINYINT
- VARCHAR

See <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types>.

10.2.38.3.5.2 Configuring Hive with a Remote Metastore Database

You can find a list of supported databases that you can use to configure remote Hive metastore can be found at <https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin#AdminManualMetastoreAdmin-SupportedBackendDatabasesforMetastore>.

The following example shows a MySQL database is configured as the Hive metastore using properties in the \${HIVE_HOME}/conf/hive-site.xml Hive configuration file.

Note

The ConnectionURL and driver class used in this example are specific to MySQL database. If you use a database other than MySQL, then change the values to fit your configuration.

```
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:mysql://MYSQL_DB_IP:MYSQL_DB_PORT/DB_NAME?
createDatabaseIfNotExist=false</value>
</property>

<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
```

```

        <value>com.mysql.jdbc.Driver</value>
    </property>

    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>MYSQL_CONNECTION_USERNAME</value>
    </property>

    <property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>MYSQL_CONNECTION_PASSWORD</value>
    </property>

```

To see a list of parameters to configure in the `hive-site.xml` file for a remote metastore, see <https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin#AdminManualMetastoreAdmin-RemoteMetastoreDatabase>.

Note

Follow these steps to add the MySQL JDBC connector JAR in the Hive classpath:

1. In `HIVE_HOME/lib/` directory, `DB_NAME` should be replaced by a valid database name created in MySQL.
2. Start the Hive Server:
`HIVE_HOME/bin/hiveserver2/bin/hiveserver2`
3. Start the Hive Remote Metastore Server:
`HIVE_HOME/bin/hive --service metastore`

10.2.38.3.5.3 Classpath Configuration

For the Hive Metadata Provider to connect to Hive, you must configure the `hive-site.xml` file and the Hive and HDFS client jars in the `gg.classpath` variable. The client JARs must match the version of Hive to which the Hive Metadata Provider is connecting.

For example, if the `hive-site.xml` file is created in the `/home/user/oggadp/dirprm` directory, then `gg.classpath` entry is `gg.classpath=/home/user/oggadp/dirprm/`

1. Create a `hive-site.xml` file that has the following properties:

```

<configuration>
<!-- Mandatory Property -->
<property>
<name>hive.metastore.uris</name>
<value>thrift://HIVE_SERVER_HOST_IP:9083</value>
</property>

<!-- Optional Property. Default value is 5 -->
<property>
<name>hive.metastore.connect.retries</name>
<value>3</value>
</property>

<!-- Optional Property. Default value is 1 -->
<property>
<name>hive.metastore.client.connect.retry.delay</name>
<value>10</value>

```

```

</property>

<!-- Optional Property. Default value is 600 seconds -->
<property>
<name>hive.metastore.client.socket.timeout</name>
<value>50</value>
</property>

</configuration>

```

2. By default, the following directories contain the Hive and HDFS client jars:

```

HIVE_HOME/hcatalog/share/hcatalog/*
HIVE_HOME/lib/*
HIVE_HOME/hcatalog/share/webhcat/java-client/*
HADOOP_HOME/share/hadoop/common/*
HADOOP_HOME/share/hadoop/common/lib/*
HADOOP_HOME/share/hadoop/mapreduce/*

```

Configure the `gg.classpath` exactly as shown in the step 1. The path to the `hive-site.xml` file must be the path with no wildcard appended. If you include the `*` wildcard in the path to the `hive-site.xml` file, it will not be located. The path to the dependency JARs must include the `*` wildcard character to include all of the JAR files in that directory in the associated classpath. Do *not* use `*.jar`.

10.2.38.3.5.4 Hive Metadata Provider Configuration Properties

Property	Required/Optional	Legal Values	Default	Explanation
<code>gg.mdp.type</code>	Required	hive	-	Selects the Hive Metadata Provider
<code>gg.mdp.connectionUrl</code>	Required	Format without Kerberos Authentication: jdbc:hive2:// HIVE_SERVER_IP:HIVE_JDBC_PORT/HIVE_DB Format with Kerberos Authentication: jdbc:hive2:// HIVE_SERVER_IP:HIVE_JDBC_PORT/HIVE_DB; principal=user/ FQDN@MY.REALM	-	The JDBC connection URL of the Hive server
<code>gg.mdp.driverClassName</code>	Required	org.apache.hive.jdbc.HiveDriver	-	The fully qualified Hive JDBC driver class name

Property	Required/ Optional	Legal Values	Default	Explanation
gg.mdp.userName	Optional	Valid username	""	The user name for connecting to the Hive database. The userName property is not required when Kerberos authentication is used. The Kerberos principal should be specified in the connection URL as specified in connectionUrl property's legal values.
gg.mdp.password	Optional	Valid Password	""	The password for connecting to the Hive database
gg.mdp.charset	Optional	Valid character set	UTF-8	The character set of the column with the character data type. Used to convert the source data from the trail file to the correct target character set.
gg.mdp.nationalCharset	Optional	Valid character set	UTF-8	The character set of the column with the national character data type. Used to convert the source data from the trail file to the correct target character set. For example, this property may indicate the character set of columns, such as NCHAR and NVARCHAR in an Oracle database.
gg.mdp.authType	Optional	Kerberos	none	Allows you to designate Kerberos authentication to Hive.
gg.mdp.kerberosKeytabFile	Optional (Required if authType=kerberos)	Relative or absolute path to a Kerberos keytab file.	-	The keytab file allows Hive to access a password to perform the kinit operation for Kerberos security.
gg.mdp.kerberosPrincipal	Optional (Required if authType=kerberos)	A legal Kerberos principal name(user/FQDN@MY.REALM)	-	The Kerberos principal name for Kerberos authentication.

10.2.38.3.5.5 Review a Sample Configuration

This is an example for configuring the Hive Metadata Provider. Consider a source with following table:

```
TABLE GG.TCUSTMER {
  CUST_CODE VARCHAR(4)    PRIMARY KEY,
  NAME VARCHAR(100),
```

```
CITY VARCHAR(200),
STATE VARCHAR(200)}
```

The example maps the column CUST_CODE (GG.TCUSTMER) in the source to CUST_CODE2 (GG_HIVE.TCUSTMER_HIVE) on the target and column CITY (GG.TCUSTMER) in the source to CITY2 (GG_HIVE.TCUSTMER_HIVE) on the target.

Mapping configuration in the *process_name*.prm file includes the following configuration:

```
MAP GG.TCUSTMER, TARGET GG_HIVE.TCUSTMER_HIVE, COLMAP(USEDEFAULTS, CUST_CODE2=CUST_CODE,
CITY2=CITY) KEYCOLS(CUST_CODE2);
```

In this example:

- The source schema GG is mapped to the target schema GG_HIVE.
- The source column CUST_CODE is mapped to the target column CUST_CODE2.
- The source column CITY is mapped to the target column CITY2.
- USEDEFAULTS specifies that rest of the column names are same on both source and target (NAME and STATE columns).
- KEYCOLS is used to specify that CUST_CODE2 should be treated as primary key.

Because primary keys cannot be specified in the Hive DDL, the KEYCOLS parameter is used to specify the primary keys.

Note

You can choose any schema name and are not restricted to the gg_hive schema name. The Hive schema can be pre-existing or newly created. You do this by modifying the connection URL (gg.mdp.connectionUrl) in the Java Adapter properties file and the mapping configuration in the Replicat.prm file. Once the schema name is changed, update the connection URL (gg.mdp.connectionUrl) and mapping in the Replicat.prm file.

You can create the schema and tables for this example in Hive by using the following commands. You can create the schema and tables for this example in Hive by using the following commands. To start the Hive CLI use the following command:

```
HIVE_HOME/bin/hive
```

To create the GG_HIVE schema, in Hive, use the following command:

```
hive> create schema gg_hive;
OK
Time taken: 0.02 seconds
```

To create the TCUSTMER_HIVE table in the GG_HIVE database, use the following command:

```
hive> CREATE EXTERNAL TABLE `TCUSTMER_HIVE` (
  > "CUST_CODE2" VARCHAR(4),
  > "NAME" VARCHAR(30),
  > "CITY2" VARCHAR(20),
  > "STATE" STRING);
OK
Time taken: 0.056 seconds
```

Configure the .properties file in a way that resembles the following:

```
gg.mdp.type=hive
gg.mdp.connectionUrl=jdbc:hive2://HIVE_SERVER_IP:10000/gg_hive
gg.mdp.driverClassName=org.apache.hive.jdbc.HiveDriver
```

The following sample output uses the delimited text formatter, with a comma as the delimiter:

```
I;GG_HIVE.TCUSTMER_HIVE;2015-10-07T04:50:47.519000;cust_code2;WILL;name;BG SOFTWARE
C0;city2;SEATTLE;state;WA
```

A sample Replicat configuration file, Java Adapter properties file, and Hive create table SQL script are included with the installation at the following location:

GoldenGate_install_directory/AdapterExamples/big-data/metadata_provider/hive

10.2.38.3.5.6 Security

You can secure the Hive server using Kerberos authentication. For information about how to secure the Hive server, see the Hive documentation for the specific Hive release. The Hive Metadata Provider can connect to a Kerberos secured Hive server.

Make sure that the paths to the HDFS core-site.xml file and the hive-site.xml file are in the handler's classpath.

Enable the following properties in the core-site.xml file:

```
<property>
<name>hadoop.security.authentication</name>
<value>kerberos</value>
</property>
```

```
<property>
<name>hadoop.security.authorization</name>
<value>true</value>
</property>
```

Enable the following properties in the hive-site.xml file:

```
<property>
<name>hive.metastore.sasl.enabled</name>
<value>true</value>
</property>
```

```
<property>
<name>hive.metastore.kerberos.keytab.file</name>
<value>/path/to/keytab</value> <!-- Change this value -->
</property>
```

```
<property>
<name>hive.metastore.kerberos.principal</name>
<value>Kerberos Principal</value> <!-- Change this value -->
</property>
```

```
<property>
  <name>hive.server2.authentication</name>
  <value>KERBEROS</value>
</property>
```

```
<property>
  <name>hive.server2.authentication.kerberos.principal</name>
  <value>Kerberos Principal</value> <!-- Change this value -->
</property>
```

```
<property>
  <name>hive.server2.authentication.kerberos.keytab</name>
  <value>/path/to/keytab</value> <!-- Change this value -->
</property>
```

10.2.38.3.5.7 Metadata Change Event

Tables in Hive metastore should be updated, altered, or created manually if the source database tables change. In the case of a metadata change event, you may wish to terminate or suspend the Replicat process. You can terminate the Replicat process by adding the following to the Replicat configuration file (*process_name.prm*):

```
DDL INCLUDE ALL, EVENTACTIONS (ABORT)
```

You can suspend the Replicat process by adding the following to the Replication configuration file:

```
DDL INCLUDE ALL, EVENTACTIONS (SUSPEND)
```

10.2.38.3.5.8 Limitations

Columns with binary data type cannot be used as primary keys.

The source-to-target mapping that is defined in the Replicat configuration file is static. Oracle GoldenGate 12.2 and later versions supports DDL propagation and source schema evolution for Oracle databases as replication sources. If you use DDL propagation and source schema evolution, you lose the ability to seamlessly handle changes to the source metadata.

10.2.38.3.5.9 Additional Considerations

The most common problems encountered are the Java classpath issues. The Hive Metadata Provider requires certain Hive and HDFS client libraries to be resolved in its classpath.

The required client JAR directories are listed in [Classpath Configuration](#). Hive and HDFS client JARs do not ship with Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA). The client JARs should be of the same version as the Hive version to which the Hive Metadata Provider is connecting.

To establish a connection to the Hive server, the *hive-site.xml* file must be in the classpath.

10.2.38.3.5.10 Troubleshooting

If the mapped target table is not present in Hive, the Replicat process will terminate with a "Table metadata resolution exception".

For example, consider the following mapping:

```
MAP GG.TCUSTMER, TARGET GG_HIVE.TCUSTMER_HIVE, COLMAP(USEDEFAULTS, CUST_CODE2=CUST_CODE,
CITY2=CITY) KEYCOLS(CUST_CODE2);
```

This mapping requires a table called *TCUSTMER_HIVE* to be created in the schema *GG_HIVE* in the Hive metastore. If this table is not present in Hive, then the following exception occurs:

```
ERROR [main] - Table Metadata Resolution Exception
Unable to retrieve table metadata. Table : GG_HIVE.TCUSTMER_HIVE
NoSuchObjectException(message:GG_HIVE.TCUSTMER_HIVE table not found)
```

10.2.38.3.6 Google BigQuery Metadata Provider

Google metadata provider uses the Google Query Job to retrieve the metadata schema information from the Google BigQuery Table. The Table should already be created on the target for BigQuery to fetch the metadata.

Google BigQuery does not support primary key semantics, so the metadata retrieved from BigQuery Table does not include any primary key definition. You can identify the primary keys using the KEYCOLS syntax in the replicat mapping statement. If KEYCOLS is not present, then the key information from the source table is used.

- [Authentication](#)
- [Supported BigQuery Datatypes](#)
- [Parameterized BigQuery Datatypes](#)
The BigQuery datatypes that can be parameterized to add constraints are STRING, BYTES, NUMERIC, and BIGNUMERIC. The STRING and BYTES datatypes can have length constraints. NUMERIC and BIGNUMERIC can have scale and precision constraints.
- [Unsupported BigQuery Datatypes](#)
- [Configuring BigQuery Metadata Provider](#)
- [Sample Configuration](#)
- [Proxy Settings](#)
- [Classpath Settings](#)
- [Limitations](#)

10.2.38.3.6.1 Authentication

Google BigQuery cloud service account can be connected either using the credentials JSON file by setting the path to the file in MDP property or setting the individual keys of credentials JSON into BigQuery MDP properties. The individual properties of BigQuery metadata provider for configuring the service account credential keys can be encrypted using Oracle wallet.

10.2.38.3.6.2 Supported BigQuery Datatypes

The following table lists the Google BigQuery datatypes that are supported and their default scale and precision values:

Data Type	Range	Max Scale	Max Precision	Max Bytes
BOOL	TRUE FALSE NIL	NA	NA	1
INT64	$[-2^{64}]$ to $[+2^{64}-1]$	NA	NA	8
FLOAT64	NA	NA	None	8

Data Type	Range	Max Scale	Max Precision	Max Bytes
NUMERIC	Min: 9.999999999 9999999999 9999999999 999999E+28 Max: 9.999999999 9999999999 9999999999 999999E+28	9	38	64
BIG NUMERIC	Min: 5.789604461 86580977117 85492504343 9539266 34992332820 28201972879 20039565648 19968E+38 Max: 5.789604461 86580977117 85492504343 9539266 34992332820 28201972879 20039565648 19967E+38	38	77	255
STRING	Unlimited	NA	NA	2147483647L
BYTES	Unlimited	NA	NA	2147483647L
DATE	0001-01-01 to 9999-12-31	NA	NA	NA
TIME	00:00:00 to 23:59:59.99 9999	NA	NA	NA
TIMESTAMP	0001-01-01 00:00:00 to 9999-12-31 23:59:59.99 9999 UTC	NA	NA	NA

10.2.38.3.6.3 Parameterized BigQuery Datatypes

The BigQuery datatypes that can be parameterized to add constraints are STRING, BYTES, NUMERIC, and BIGNUMERIC. The STRING and BYTES datatypes can have length constraints. NUMERIC and BIGNUMERIC can have scale and precision constraints.

1. STRING(L): L is the maximum number of Unicode characters allowed.
2. BYTES(L): L is the maximum number of bytes allowed.

3. NUMERIC(P[, S]) or BIGNUMERIC(P[, S]): P is maximum precision (total number of digits) and S is maximum scale (number of digits after decimal) that is allowed.

The parameterized datatypes are supported in BigQuery Metadata Provider. If there is a datatype with user-defined precision, scale or max-length, then metadata provider calculates the data based on those values.

10.2.38.3.6.4 Unsupported BigQuery Datatypes

The following table lists the Google BigQuery datatypes that are supported and their default scale and precision values:

The BigQuery datatypes that are not supported by metadata provider are complex datatypes, such as GEOGRAPHY, JSON, ARRAY, INTERVAL, and STRUCT. The metadata provider is going to abend with invalid datatype exception if it encounters them.

10.2.38.3.6.5 Configuring BigQuery Metadata Provider

The following table lists the configuration properties for BigQuery metadata provider:

Property	Required/ Optional	Legal Values	Default	Explanation
gg.mdp.type	Required	bq	NA	Select BigQuery Metadata Provider
gg.mdp.credentialsFile	Optional	File path to credentials JSON file.	NA	Provides path to the credentials JSON file for connecting to Google BigQuery Service account.
gg.mdp.clientId	Optional	Valid BigQuery Credentials Client Id	NA	Provides the client Id key from the credentials file for connecting to Google BigQuery service account.
gg.mdp.clientEmail	Optional	Valid BigQuery Credentials Client Email	NA	Provides the client Email key from the credentials file for connecting to Google BigQuery service account.
gg.mdp.privateKeyId	Optional	Valid BigQuery Credentials Private Key ID	NA	Provides the Private Key ID from the credentials file for connecting to Google BigQuery service account.
gg.mdp.privateKey	Optional	Valid BigQuery Credentials Private Key	NA	Provides the Private Key from the credentials file for connecting to Google BigQuery service account.

Property	Required/ Optional	Legal Values	Default	Explanations
gg.mdp.projectId	Optional	Unique BigQuery project Id	NA	Unique project Id of BigQuery.
gg.mdp.connectionTimeout	Optional	Time in sec	5	Connect Timeout for BigQuery connection.
gg.mdp.readTimeout	Optional	Time in sec	6	Timeout to read from BigQuery connection.
gg.mdp.totalTimeout	Optional	Time in sec	9	Total timeout for BigQuery connection.
gg.mdp.retryCount	Optional	Maximum number of retries.	3	Maximum number of retries for connecting to BigQuery.

Either of the property to set the path to credentials JSON file or the properties to set the credential file keys are mandatory for connecting to Google Service account for accessing the BigQuery. Setting the individual credentials parameter enables them to be encrypted using Oracle wallet.

10.2.38.3.6.6 Sample Configuration

Sample properties file content:

The following are sample properties that are added to BigQuery Handler properties file or BigQuery Event Handler properties file along with their own properties in order to configure the metadata provider.

```
gg.mdp.type=bq
gg.mdp.credentialsFile=/path/to/credFile.json
```

Sample parameter file:

There is no change in parameter file for configuring metadata provider. This sample parameter file is similar to the BigQuery Event Handler parameter file.

```
REPLICAT bqeh
TARGETDB LIBFILE libggjava.so SET property=dirprm/bqeh.props
MAP schema.tableName, TARGET schema.tableName;
```

10.2.38.3.6.7 Proxy Settings

The proxy settings can be added as java virtual machine (JVM) arguments when we are trying to access the BigQuery server from behind a proxy. For example, for oracle proxy server connection can be added in properties file as follows:

```
jvm.bootoptions= -Dhttps.proxyHost=www-proxy.us.oracle.com -
Dhttps.proxyPort=80
```


10.2.38.3.6.8 Classpath Settings

The dependency of BigQuery metadata provider is same as the Google BigQuery stage-and-merge Event Handler dependency. The dependencies added to the Oracle GoldenGate classpath for BigQuery event Handler is sufficient for running the BigQuery metadata provider, and no extra dependency need to be configured.

10.2.38.3.6.9 Limitations

The complex BigQuery datatypes are not yet supported by the metadata provider. It will abend in case any of unsupported datatypes are encountered.

If the BigQuery handler or event-handler is configured to auto create table and dataspace, then the metadata provider expects table to exist in order to fetch the metadata. The feature to auto-create table and dataspace of BigQuery handler and event handler does not work with BigQuery metadata provider. Metadata change event is not supported by Big Query metadata provider. It can be configured to abend or suspend in case there is a metadata change.

10.2.38.4 Pluggable Formatters

The pluggable formatters are used to convert operations from the Oracle GoldenGate trail file into formatted messages that you can send to Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) targets using one of the GG for DAA handlers.

This chapter describes how to use the pluggable formatters.

- [Using Operation-Based versus Row-Based Formatting](#)
The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) formatters include operation-based and row-based formatters.
- [Using the Avro Formatter](#)
Apache Avro is an open source data serialization and deserialization framework known for its flexibility, compactness of serialized data, and good serialization and deserialization performance. Apache Avro is commonly used in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) applications.
- [Cloud Event Formatter](#)
- [Existing Avro Formatter](#)
- [Using the Delimited Text Formatter](#)
- [Using the JSON Formatter](#)
- [Using the Length Delimited Value Formatter](#)
The Length Delimited Value (LDV) Formatter is a row-based formatter. It formats database operations from the source trail file into a length delimited value output. Each insert, update, delete, or truncate operation from the source trail is formatted into an individual length delimited message.
- [Using the XML Formatter](#)
The XML Formatter formats before-image and after-image data from the source trail file into an XML document representation of the operation data. The format of the XML document is effectively the same as the XML format in the previous releases of the Oracle GoldenGate Java Adapter.

10.2.38.4.1 Using Operation-Based versus Row-Based Formatting

The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) formatters include operation-based and row-based formatters.

The operation-based formatters represent the individual insert, update, and delete events that occur on table data in the source database. Insert operations only provide after-change data (or images), because a new row is being added to the source database. Update operations provide both before-change and after-change data that shows how existing row data is modified. Delete operations only provide before-change data to identify the row being deleted. The operation-based formatters model the operation as it exists in the source trail file. Operation-based formats include fields for the before-change and after-change images.

The row-based formatters model the row data as it exists after the operation data is applied. Row-based formatters contain only a single image of the data. The following sections describe what data is displayed for both the operation-based and the row-based formatters.

- [Operation Formatters](#)
- [Row Formatters](#)
- [Table Row or Column Value States](#)

10.2.38.4.1.1 Operation Formatters

The formatters that support operation-based formatting are JSON, Avro Operation, and XML. The output of operation-based formatters are as follows:

- Insert operation: Before-image data is null. After image data is output.
- Update operation: Both before-image and after-image data is output.
- Delete operation: Before-image data is output. After-image data is null.
- Truncate operation: Both before-image and after-image data is null.

10.2.38.4.1.2 Row Formatters

The formatters that support row-based formatting are Delimited Text and Avro Row. Row-based formatters output the following information for the following operations:

- Insert operation: After-image data only.
- Update operation: After-image data only. Primary key updates are a special case which will be discussed in individual sections for the specific formatters.
- Delete operation: Before-image data only.
- Truncate operation: The table name is provided, but both before-image and after-image data are null. Truncate table is a DDL operation, and it may not support different database implementations. Refer to the *Oracle GoldenGate* documentation for your database implementation.

10.2.38.4.1.3 Table Row or Column Value States

In an RDBMS, table data for a specific row and column can only have one of two states: either the data has a value, or it is null. However; when data is transferred to the Oracle GoldenGate trail file by the Oracle GoldenGate capture process, the data can have three possible states: it can have a value, it can be null, or it can be missing.

For an insert operation, the after-image contains data for all column values regardless of whether the data is null. However, the data included for update and delete operations may not always contain complete data for all columns. When replicating data to an RDBMS for an update operation only the primary key values and the values of the columns that changed are required to modify the data in the target database. In addition, only the primary key values are required to delete the row from the target database. Therefore, even though values are present in the source database, the values may be missing in the source trail file. Because data in the source trail file may have three states, the Plugable Formatters must also be able to represent data in all three states.

Because the row and column data in the Oracle GoldenGate trail file has an important effect on Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) integration, it is important to understand the data that is required. Typically, you can control the data that is included for operations in the Oracle GoldenGate trail file. In an Oracle database, this data is controlled by the supplemental logging level. To understand how to control the row and column values that are included in the Oracle GoldenGate trail file, see the *Oracle GoldenGate* documentation for your source database implementation..

10.2.38.4.2 Using the Avro Formatter

Apache Avro is an open source data serialization and deserialization framework known for its flexibility, compactness of serialized data, and good serialization and deserialization performance. Apache Avro is commonly used in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) applications.

- [Avro Row Formatter](#)
- [The Avro Operation Formatter](#)
- [Avro Object Container File Formatter](#)

10.2.38.4.2.1 Avro Row Formatter

The Avro Row Formatter formats operation data from the source trail file into messages in an Avro binary array format. Each individual insert, update, delete, and truncate operation is formatted into an individual Avro message. The source trail file contains the before and after images of the operation data. The Avro Row Formatter takes the before-image and after-image data and formats it into an Avro binary representation of the operation data.

The Avro Row Formatter formats operations from the source trail file into a format that represents the row data. This format is more compact than the output from the Avro Operation Formatter for the Avro messages model the change data operation.

The Avro Row Formatter may be a good choice when streaming Avro data to HDFS. Hive supports data files in HDFS in an Avro format.

This section contains the following topics:

- [Operation Metadata Formatting Details](#)
The automated output of meta-column fields in generated Avro messages has been removed as of Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) release 21.1. Meta-column fields can still be output; however, they need to explicitly configured as the following property: `gg.handler.name.format.metaColumnsTemplate`.
- [Operation Data Formatting Details](#)
- [Sample Avro Row Messages](#)
- [Avro Schemas](#)
Avro uses JSONs to represent schemas. Avro schemas define the format of generated Avro messages and are required to serialize and deserialize Avro messages.

- [Avro Row Configuration Properties](#)
- [Review a Sample Configuration](#)
- [Metadata Change Events](#)
- [Special Considerations](#)

10.2.38.4.2.1.1 Operation Metadata Formatting Details

The automated output of meta-column fields in generated Avro messages has been removed as of Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) release 21.1. Meta-column fields can still be output; however, they need to be explicitly configured as the following property: `gg.handler.name.format.metaColumnsTemplate`.

To output the metacolumns configure the following:

```
gg.handler.name.format.metaColumnsTemplate=${objectname[table]},${
{optype[op_type]},${timestamp[op_ts]},${currenttimestamp[current_ts]},$
{position[pos]}}
```

To also include the primary key columns and the tokens configure as follows:

```
gg.handler.name.format.metaColumnsTemplate=${objectname[table]},$
{optype[op_type]},${timestamp[op_ts]},${currenttimestamp[current_ts]},$
{position[pos]},${primarykeycolumns[primary_keys]},${alltokens[tokens]}
```

For more information see the configuration property:

`gg.handler.name.format.metaColumnsTemplate`.

Table 10-50 Avro Formatter Metadata

Value	Description
table	The fully qualified table in the format is: <i>CATALOG_NAME . SCHEMA_NAME . TABLE_NAME</i>
op_type	The type of database operation from the source trail file. Default values are I for insert, U for update, D for delete, and T for truncate.
op_ts	The timestamp of the operation from the source trail file. Since this timestamp is from the source trail, it is fixed. Replaying the trail file results in the same timestamp for the same operation.
current_ts	The time when the formatter processed the current operation record. This timestamp follows the ISO-8601 format and includes microsecond precision. Replaying the trail file will <i>not</i> result in the same timestamp for the same operation.
pos	The concatenated sequence number and the RBA number from the source trail file. This trail position lets you trace the operation back to the source trail file. The sequence number is the source trail file number. The RBA number is the offset in the trail file.
primary_keys	An array variable that holds the column names of the primary keys of the source table.
tokens	A map variable that holds the token key value pairs from the source trail file.

10.2.38.4.2.1.2 Operation Data Formatting Details

The operation data follows the operation metadata. This data is represented as individual fields identified by the column names.

Column values for an operation from the source trail file can have one of three states: the column has a value, the column value is null, or the column value is missing. Avro attributes only support two states, the column has a value or the column value is null. Missing column values are handled the same as null values. Oracle recommends that when you use the Avro Row Formatter, you configure the Oracle GoldenGate capture process to provide full image data for all columns in the source trail file.

By default, the setting of the Avro Row Formatter maps the data types from the source trail file to the associated Avro data type. Because Avro provides limited support for data types, source columns map into Avro long, double, float, binary, or string data types. You can also configure data type mapping to handle all data as strings.

10.2.38.4.2.1.3 Sample Avro Row Messages

Because Avro messages are binary, they are not human readable. The following sample messages show the JSON representation of the messages.

- [Sample Insert Message](#)
- [Sample Update Message](#)
- [Sample Delete Message](#)
- [Sample Truncate Message](#)

10.2.38.4.2.1.3.1 Sample Insert Message

```
{
  "table": "GG.TCUSTORD",
  "op_type": "I",
  "op_ts": "2013-06-02 22:14:36.000000",
  "current_ts": "2015-09-18T10:13:11.172000",
  "pos": "00000000000000001444",
  "primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"],
  "tokens": {"R": "AADPkVAAEAAEqL2AAA"},
  "CUST_CODE": "WILL",
  "ORDER_DATE": "1994-09-30:15:33:00",
  "PRODUCT_CODE": "CAR",
  "ORDER_ID": "144",
  "PRODUCT_PRICE": 17520.0,
  "PRODUCT_AMOUNT": 3.0,
  "TRANSACTION_ID": "100"}

```

10.2.38.4.2.1.3.2 Sample Update Message

```
{
  "table": "GG.TCUSTORD",
  "op_type": "U",
  "op_ts": "2013-06-02 22:14:41.000000",
  "current_ts": "2015-09-18T10:13:11.492000",
  "pos": "00000000000000002891",
  "primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"],
  "tokens": {"R": "AADPkVAAEAAEqLzAAA"},
  "CUST_CODE": "BILL",
  "ORDER_DATE": "1995-12-31:15:00:00",
  "PRODUCT_CODE": "CAR",
  "ORDER_ID": "765",
  "PRODUCT_PRICE": 14000.0,
  "PRODUCT_AMOUNT": 3.0,
  "TRANSACTION_ID": "100"}

```

10.2.38.4.2.1.3.3 Sample Delete Message

```
{
  "table": "GG.TCUSTORD",
  "op_type": "D",
  "op_ts": "2013-06-02 22:14:41.000000",

```

```
"current_ts": "2015-09-18T10:13:11.512000",
"pos": "00000000000000004338",
"primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"], "tokens":
{"L": "206080450", "6": "9.0.80330", "R": "AADPkVAAEAAEqLZAAC"}, "CUST_CODE":
"DAVE",
"ORDER_DATE": "1993-11-03:07:51:35",
"PRODUCT_CODE": "PLANE",
"ORDER_ID": "600",
"PRODUCT_PRICE": null,
"PRODUCT_AMOUNT": null,
"TRANSACTION_ID": null}
```

10.2.38.4.2.1.3.4 Sample Truncate Message

```
{"table": "GG.TCUSTORD",
"op_type": "T",
"op_ts": "2013-06-02 22:14:41.000000",
"current_ts": "2015-09-18T10:13:11.514000",
"pos": "00000000000000004515",
"primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"], "tokens":
{"R": "AADPkVAAEAAEqL2AAB"},
"CUST_CODE": null,
"ORDER_DATE": null,
"PRODUCT_CODE": null,
"ORDER_ID": null,
"PRODUCT_PRICE": null,
"PRODUCT_AMOUNT": null,
"TRANSACTION_ID": null}
```

10.2.38.4.2.1.4 Avro Schemas

Avro uses JSONs to represent schemas. Avro schemas define the format of generated Avro messages and are required to serialize and deserialize Avro messages.

Schemas are generated on a just-in-time basis when the first operation for a table is encountered. Newer schemas are generated when there is a change in the metadata. The generated Avro schemas are specific to a table definition, and therefore, a separate Avro schema is generated for every table encountered for processed operations. By default, Avro schemas are written to the *GoldenGate_Home/dirdef* directory, although the write location is configurable. Avro schema file names adhere to the following naming convention:
Fully_Qualified_Table_Name.avsc.

The following is a sample Avro schema for the Avro Row Format for the references examples in the previous section:

```
{
  "type" : "record",
  "name" : "TCUSTORD",
  "namespace" : "GG",
  "fields" : [ {
    "name" : "table",
    "type" : "string"
  }, {
    "name" : "op_type",
    "type" : "string"
  }, {
    "name" : "op_ts",
    "type" : "string"
  }, {
    "name" : "current_ts",
    "type" : "string"
  }, {
    "name" : "pos",
```

```

    "type" : "string"
  }, {
    "name" : "primary_keys",
    "type" : {
      "type" : "array",
      "items" : "string"
    }
  }, {
    "name" : "tokens",
    "type" : {
      "type" : "map",
      "values" : "string"
    },
    "default" : { }
  }, {
    "name" : "CUST_CODE",
    "type" : [ "null", "string" ],
    "default" : null
  }, {
    "name" : "ORDER_DATE",
    "type" : [ "null", "string" ],
    "default" : null
  }, {
    "name" : "PRODUCT_CODE",
    "type" : [ "null", "string" ],
    "default" : null
  }, {
    "name" : "ORDER_ID",
    "type" : [ "null", "string" ],
    "default" : null
  }, {
    "name" : "PRODUCT_PRICE",
    "type" : [ "null", "double" ],
    "default" : null
  }, {
    "name" : "PRODUCT_AMOUNT",
    "type" : [ "null", "double" ],
    "default" : null
  }, {
    "name" : "TRANSACTION_ID",
    "type" : [ "null", "string" ],
    "default" : null
  } ]
}

```

10.2.38.4.2.1.5 Avro Row Configuration Properties

Table 10-51 Avro Row Configuration Properties

Properties	Optional/Required	Legal Values	Default	Explanation
gg.handler.name.format.insertOpKey	Optional	Any string	I	Indicator to be inserted into the output record to indicate an insert operation.

Table 10-51 (Cont.) Avro Row Configuration Properties

Properties	Optional/Required	Legal Values	Default	Explanation
<code>gg.handler.name.format.updateOpKey</code>	Optional	Any string	U	Indicator to be inserted into the output record to indicate an update operation.
<code>gg.handler.name.format.deleteOpKey</code>	Optional	Any string	D	Indicator to be inserted into the output record to indicate a delete operation.
<code>gg.handler.name.format.truncateOpKey</code>	Optional	Any string	T	Indicator to be inserted into the output record to indicate a truncate operation.
<code>gg.handler.name.format.encoding</code>	Optional	Any legal encoding name or alias supported by Java.	UTF-8 (the JSON default is UTF-8. Avro messages are binary and support their own internal representation of encoding).	Controls the output encoding of generated JSON Avro schema. The JSON default is UTF-8. Avro messages are binary and support their own internal representation of encoding.
<code>gg.handler.name.format.treatAllColumnsAsStrings</code>	Optional	true false	false	Controls the output typing of generated Avro messages. If set to false then the formatter will attempt to map Oracle GoldenGate types to the corresponding AVRO type. If set to true then all data will be treated as Strings in the generated Avro messages and schemas.

Table 10-51 (Cont.) Avro Row Configuration Properties

Properties	Optional/Required	Legal Values	Default	Explanation
<code>gg.handler.name.format.pkUpdateHandling</code>	Optional	<code>abend</code> <code>update</code> <code>delete-insert</code>	<code>abend</code>	Specifies how the formatter handles update operations that change a primary key. Primary key operations for the Avro Row formatter require special consideration. <code>abend</code>: the process terminates. <code>update</code>: the process handles the update as a normal update. <code>delete - insert</code>: the process handles the update as a delete and an insert. Full supplemental logging must be enabled. Without full before and after row images, the insert data will be incomplete.
<code>gg.handler.name.format.lineDelimiter</code>	Optional	Any string	no value	Inserts a delimiter after each Avro message. This is not a best practice, but in certain cases you may want to parse a stream of data and extract individual Avro messages from the stream. Select a unique delimiter that cannot occur in any Avro message. This property supports CDATA[] wrapping.

Table 10-51 (Cont.) Avro Row Configuration Properties

Properties	Optional/Required	Legal Values	Default	Explanation
<code>gg.handler.name.format.versionSchemas</code>	Optional	true false	false	Avro schemas always follow the <i>fully_qualified_table_name.avsc</i> convention. Setting this property to true creates an additional Avro schema named <i>fully_qualified_table_name_current_timestamp.avsc</i> in the schema directory. Because the additional Avro schema is not destroyed or removed, provides a history of schema evolution.
<code>gg.handler.name.format.wrapMessageInGenericAvroMessage</code>	Optional	true false	false	Wraps the Avro messages for operations from the source trail file in a generic Avro wrapper message. For more information, see Generic Wrapper Functionality .
<code>gg.handler.name.format.schemaDirectory</code>	Optional	Any legal, existing file system path.	./dir	The output location of generated Avro schemas.
<code>gg.handler.name.format.schemaFilePath</code>	Optional	Any legal encoding name or alias supported by Java.	./dir	The directory in the HDFS where schemas are output. A metadata change overwrites the schema during the next operation for the associated table. Schemas follow the same naming convention as schemas written to the local file system: <i>catalog.schema.table.avsc</i> .

Table 10-51 (Cont.) Avro Row Configuration Properties

Properties	Optional/ Required	Legal Values	Default	Explanation
<code>gg.handler.name.format.iso8601Format</code>	Optional	true false	true	The format of the current timestamp. The default is the ISO 8601 format. A setting of false removes the T between the date and time in the current timestamp, which outputs a space instead.
<code>gg.handler.name.format.includeIsMissingFields</code>	Optional	true false	false	Set to true to include a <code>{column_name}_isMissing</code> boolean field for each source field. This field allows downstream applications to differentiate if a null value is null in the source trail file (value is false) or is missing in the source trail file (value is true).
<code>gg.handler.name.format.enableDecimalLogicalType</code>	Optional	true false	false	Enables the use of Avro decimal logical types. The decimal logical type represents numbers as a byte array and can provide support for much larger numbers than can fit in the classic 64-bit long or double data types.

Table 10-51 (Cont.) Avro Row Configuration Properties

Properties	Optional/ Required	Legal Values	Default	Explanation
<code>gg.handler.name.format.oracleNumberScale</code>	Optional	Any integer value from 0 to 38.	None	Allows you to set the scale on the Avro decimal data type. Only applicable when you set <code>enableDecimalLogicalType=true</code> . The Oracle NUMBER is a proprietary numeric data type of Oracle Database that supports variable precision and scale. Precision and scale are variable on a per instance of the Oracle NUMBER data type. Precision and scale are required parameters when generating the Avro decimal logical type. This makes mapping of Oracle NUMBER data types into Avro difficult because there is no way to deterministically know the precision and scale of an Oracle NUMBER data type when the Avro schema is generated. The best alternative is to generate a large Avro decimal data type a precision of 164 and a scale of 38, which should hold any legal instance of Oracle NUMBER. While this solves the problem of precision loss when converting Oracle Number data types to Avro decimal data types, you may not like that Avro decimal data types when retrieved from Avro messages downstream have 38 digits trailing the decimal point.

Table 10-51 (Cont.) Avro Row Configuration Properties

Properties	Optional/ Required	Legal Values	Default	Explanation
<code>gg.handler.name.format.mapOracleNumbersAsStrings</code>	Optional	true false	false	This property is only applicable if decimal logical types are enabled via the property <code>gg.handler.name.format.enableDecimalLogicalType=true</code> . Oracle numbers are especially problematic because they have a large precision (168) and floating scale of up to 38. Some analytical tools, such as Spark cannot read numbers that large. This property allows you to map those Oracle numbers as strings while still mapping the smaller numbers as decimal logical types.
<code>gg.handler.name.format.enableTimestampLogicalType</code>	Optional	true false	false	Set to true to map source date and time data types into the Avro <code>TimestampMicros</code> logical data type. The variable <code>gg.format.timestamp</code> must be configured to provide a mask for the source date and time data types to make sense of them. The Avro <code>TimestampMicros</code> is part of the Avro 1.8 specification. If <code>gghandler.name.format.enableTimestampLogicalType</code> is set to true and <code>gg.format.timestamp</code> is unset, then replicat will abend with a configuration exception.

Table 10-51 (Cont.) Avro Row Configuration Properties

Properties	Optional/Required	Legal Values	Default	Explanation
<code>gg.handler.name.format.mapLargeNumbersAsString</code>	Optional	true false	false	Oracle GoldenGate supports the floating point and integer source datatypes. Some of these datatypes may not fit into the Avro primitive double or long datatypes. Set this property to true to map the fields that do not fit into the Avro primitive double or long datatypes to Avro string.
<code>gg.handler.name.format.metaColumnsTemplate</code>	Optional	See Metacolumn Keywords .	None	The current meta column information can be configured in a simple manner and removes the explicit need to use: <pre>insertOpKey updateOpKey deleteOpKey truncateOpKey includeTableName includeOpTimestamp includeOpType includePosition includeCurrentTimestamp, useIso8601Format</pre> It is a comma-delimited string consisting of one or more templated values that represent the template. For more information about the Metacolumn keywords, see Metacolumn Keywords.

Table 10-51 (Cont.) Avro Row Configuration Properties

Properties	Optional/Required	Legal Values	Default	Explanation
gg.handler.name.format.maxPrecision	Optional	None	Positive Integer	Allows you to set the maximum precision for Avro decimal logical types. Consuming applications may have limitations on Avro precision (that is, Apache Spark supports a maximum precision of 38).

 **Warning**
Configuration of this property is not

Table 10-51 (Cont.) Avro Row Configuration Properties

Properties	Optional/Required	Legal Values	Default	Explanation
				The NUMBER type in an Oracle RDBMS supports a maximum precision of 164. Configuration of this property likely means you are casting larger source numeric types to smaller target numeric types. If the precision of the source value is greater than the configured precision, then a runtime exception will occur and the replicat process will abend. That behavior is not a bug. That is the expected behavior.

10.2.38.4.2.1.6 Review a Sample Configuration

The following is a sample configuration for the Avro Row Formatter in the Java Adapter properties file:

```
gg.handler.hdfs.format.avro_row
gg.handler.hdfs.format.insertOpKey=I
gg.handler.hdfs.format.updateOpKey=U
gg.handler.hdfs.format.deleteOpKey=D
gg.handler.hdfs.format.truncateOpKey=T
gg.handler.hdfs.format.encoding=UTF-8
gg.handler.hdfs.format.pkUpdateHandling=abend
gg.handler.hdfs.format.wrapMessageInGenericAvroMessage=false
```

10.2.38.4.2.1.7 Metadata Change Events

If the replicated database and upstream Oracle GoldenGate replication process can propagate metadata change events, the Avro Row Formatter can take action when metadata changes. Because Avro messages depend closely on their corresponding schema, metadata changes are important when you use Avro formatting.

An updated Avro schema is generated as soon as a table operation occurs after a metadata change event. You must understand the impact of a metadata change event and change downstream targets to the new Avro schema. The tight dependency of Avro messages to Avro schemas may result in compatibility issues. Avro messages generated before the schema change may not be able to be deserialized with the newly generated Avro schema.

Conversely, Avro messages generated after the schema change may not be able to be deserialized with the previous Avro schema. It is a best practice to use the same version of the Avro schema that was used to generate the message. For more information, consult the Apache Avro documentation.

10.2.38.4.2.1.8 Special Considerations

This sections describes these special considerations:

- [Troubleshooting](#)
- [Primary Key Updates](#)
- [Generic Wrapper Functionality](#)

10.2.38.4.2.1.8.1 Troubleshooting

Because Avro is a binary format, it is not human readable. Since Avro messages are in binary format, it is difficult to debug any issue, the Avro Row Formatter provides a special feature to help debug issues. When the `log4j` Java logging level is set to `TRACE`, Avro messages are deserialized and displayed in the log file as a JSON object, letting you view the structure and contents of the created Avro messages. Do not enable `TRACE` in a production environment as it has substantial negative impact on performance. To troubleshoot content, you may want to consider switching to use a formatter that produces human-readable content. The XML or JSON formatters both produce content in human-readable format.

10.2.38.4.2.1.8.2 Primary Key Updates

In Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) integrations, primary key update operations require special consideration and planning. Primary key updates modify one or more of the primary keys of a given row in the source database. Because data is appended in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) applications, a primary key update operation looks more like a new insert than like an update without special handling. You can use the following properties to configure the Avro Row Formatter to handle primary keys:

Table 10-52 Configurable behavior

Value	Description
abend	The formatter terminates. This behavior is the default behavior.
update	With this configuration the primary key update is treated like any other update operation. Use this configuration only if you can guarantee that the primary key is not used as selection criteria row data from a GG for DAA system.
delete-insert	The primary key update is treated as a special case of a delete, using the before image data and an insert using the after-image data. This configuration may more accurately model the effect of a primary key update in a GG for DAA application. However, if this configuration is selected, it is important to have full supplemental logging enabled on Replication at the source database. Without full supplemental logging the delete operation will be correct, but insert operation will not contain all of the data for all of the columns for a full representation of the row data in the GG for DAA application.

10.2.38.4.2.1.8.3 Generic Wrapper Functionality

Because Avro messages are not self describing, the receiver of the message must know the schema associated with the message before the message can be deserialized. Avro messages are binary and provide no consistent or reliable way to inspect the message contents in order to ascertain the message type. Therefore, Avro can be troublesome when messages are interlaced into a single stream of data such as Kafka.

The Avro formatter provides a special feature to wrap the Avro message in a generic Avro message. You can enable this functionality by setting the following configuration property.

```
gg.handler.name.format.wrapMessageInGenericAvroMessage=true
```

The generic message is Avro message wrapping the Avro payload message that is common to all Avro messages that are output. The schema for the generic message is name `generic_wrapper.avsc` and is written to the output schema directory. This message has the following three fields:

- `table_name` : The fully qualified source table name.
- `schema_fingerprint` : The fingerprint of the Avro schema of the wrapped message. The fingerprint is generated using the Avro `SchemaNormalization.parsingFingerprint64(schema)` call.
- `payload`: The wrapped Avro message.

The following is the Avro Formatter generic wrapper schema.

```
{
  "type" : "record",
  "name" : "generic_wrapper",
  "namespace" : "oracle.goldengate",
  "fields" : [ {
    "name" : "table_name",
    "type" : "string"
  }, {
    "name" : "schema_fingerprint",
    "type" : "long"
  }, {
    "name" : "payload",
    "type" : "bytes"
  } ]
}
```

10.2.38.4.2.2 The Avro Operation Formatter

The Avro Operation Formatter formats operation data from the source trail file into messages in an Avro binary array format. Each individual insert, update, delete, and truncate operation is formatted into an individual Avro message. The source trail file contains the before and after images of the operation data. The Avro Operation Formatter formats this data into an Avro binary representation of the operation data.

This format is more verbose than the output of the Avro Row Formatter for which the Avro messages model the row data.

- [Operation Metadata Formatting Details](#)
- [Operation Data Formatting Details](#)
- [Sample Avro Operation Messages](#)
- [Avro Schema](#)
- [Avro Operation Formatter Configuration Properties](#)

- [Review a Sample Configuration](#)
- [Metadata Change Events](#)
- [Special Considerations](#)

10.2.38.4.2.2.1 Operation Metadata Formatting Details

To output the metacolumns configure the following:

```
gg.handler.name.format.metaColumnsTemplate=${objectname[table]},$
{optype[op_type]},${timestamp[op_ts]},${currenttimestamp[current_ts]},$
{position[pos]}
```

To also include the primary key columns and the tokens configure as follows:

```
gg.handler.name.format.metaColumnsTemplate=${objectname[table]},$
{optype[op_type]},${timestamp[op_ts]},${currenttimestamp[current_ts]},$
{position[pos]},${primarykeycolumns[primary_keys]},${alltokens[tokens]}
```

For more information see the configuration property:

```
gg.handler.name.format.metaColumnsTemplate
```

Table 10-53 Avro Messages and its Metadata

Fields	Description
table	The fully qualified table name, in the format: <i>CATALOG_NAME.SCHEMA_NAME.TABLE_NAME</i>
op_type	The type of database operation from the source trail file. Default values are I for insert, U for update, D for delete, and T for truncate.
op_ts	The timestamp of the operation from the source trail file. Since this timestamp is from the source trail, it is fixed. Replaying the trail file results in the same timestamp for the same operation.
current_ts	The time when the formatter processed the current operation record. This timestamp follows the ISO-8601 format and includes microsecond precision. Replaying the trail file will <i>not</i> result in the same timestamp for the same operation.
pos	The concatenated sequence number and rba number from the source trail file. The trail position provides traceability of the operation back to the source trail file. The sequence number is the source trail file number. The rba number is the offset in the trail file.
primary_keys	An array variable that holds the column names of the primary keys of the source table.
tokens	A map variable that holds the token key value pairs from the source trail file.

10.2.38.4.2.2.2 Operation Data Formatting Details

The operation data is represented as individual fields identified by the column names.

Column values for an operation from the source trail file can have one of three states: the column has a value, the column value is null, or the column value is missing. Avro attributes only support two states: the column has a value or the column value is null. The Avro Operation Formatter contains an additional Boolean field *COLUMN_NAME_isMissing* for each column to indicate whether the column value is missing or not. Using *COLUMN_NAME* field together with the *COLUMN_NAME_isMissing* field, all three states can be defined.

- State 1: The column has a value
COLUMN_NAME field has a value

`COLUMN_NAME_isMissing` field is false

- State 2: The column value is null

COLUMN_NAME field value is null

`COLUMN_NAME_isMissing` field is false

- State 3: The column value is missing

COLUMN_NAME field value is null

`COLUMN_NAME_isMissing` field is true

By default the Avro Row Formatter maps the data types from the source trail file to the associated Avro data type. Because Avro supports few data types, this functionality usually results in the mapping of numeric fields from the source trail file to members typed as numbers. You can also configure this data type mapping to handle all data as strings.

10.2.38.4.2.2.3 Sample Avro Operation Messages

Because Avro messages are binary, they are not human readable. The following topics show example Avro messages in JSON format:

- Sample Insert Message
- Sample Update Message
- Sample Delete Message
- Sample Truncate Message

10.2.38.4.2.2.3.1 Sample Insert Message

```
{
  "table": "GG.TCUSTORD",
  "op_type": "I",
  "op_ts": "2013-06-02 22:14:36.000000",
  "current_ts": "2015-09-18T10:17:49.570000",
  "pos": "000000000000000001444",
  "primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"], "tokens": {
    "R": "AADPkvAAEAAEqL2AAA"
  },
  "before": null,
  "after": {
    "CUST_CODE": "WILL",
    "CUST_CODE_isMissing": false,
    "ORDER_DATE": "1994-09-30:15:33:00",
    "ORDER_DATE_isMissing": false,
    "PRODUCT_CODE": "CAR",
    "PRODUCT_CODE_isMissing": false,
    "ORDER_ID": "144", "ORDER_ID_isMissing": false,
    "PRODUCT_PRICE": 17520.0,
    "PRODUCT_PRICE_isMissing": false,
    "PRODUCT_AMOUNT": 3.0, "PRODUCT_AMOUNT_isMissing": false,
    "TRANSACTION_ID": "100",
    "TRANSACTION_ID_isMissing": false
  }
}
```

10.2.38.4.2.2.3.2 Sample Update Message

```
{
  "table": "GG.TCUSTORD",
  "op_type": "U",
  "op_ts": "2013-06-02 22:14:41.000000",
  "current_ts": "2015-09-18T10:17:49.880000",
  "pos": "00000000000000002891",
  "primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"],
  "tokens": {
    "R": "AADPkvAAEAAEqLzAAA"
  },
  "before": {
    "CUST_CODE": "BILL",

```

```

"CUST_CODE_isMissing": false,
"ORDER_DATE": "1995-12-31:15:00:00",
"ORDER_DATE_isMissing": false,
"PRODUCT_CODE": "CAR",
"PRODUCT_CODE_isMissing": false,
"ORDER_ID": "765",
"ORDER_ID_isMissing": false,
"PRODUCT_PRICE": 15000.0,
"PRODUCT_PRICE_isMissing": false,
"PRODUCT_AMOUNT": 3.0,
"PRODUCT_AMOUNT_isMissing": false,
"TRANSACTION_ID": "100",
"TRANSACTION_ID_isMissing": false},
"after": {
"CUST_CODE": "BILL",
"CUST_CODE_isMissing": false,
"ORDER_DATE": "1995-12-31:15:00:00",
"ORDER_DATE_isMissing": false,
"PRODUCT_CODE": "CAR",
"PRODUCT_CODE_isMissing": false,
"ORDER_ID": "765",
"ORDER_ID_isMissing": false,
"PRODUCT_PRICE": 14000.0,
"PRODUCT_PRICE_isMissing": false,
"PRODUCT_AMOUNT": 3.0,
"PRODUCT_AMOUNT_isMissing": false,
"TRANSACTION_ID": "100",
"TRANSACTION_ID_isMissing": false}}

```

10.2.38.4.2.2.3.3 Sample Delete Message

```

{"table": "GG.TCUSTORD",
"op_type": "D",
"op_ts": "2013-06-02 22:14:41.000000",
"current_ts": "2015-09-18T10:17:49.899000",
"pos": "00000000000000004338",
"primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"], "tokens":
{"L": "206080450", "6": "9.0.80330", "R": "AADPkVAAEAeqLzAAC"}, "before": {
"CUST_CODE": "DAVE",
"CUST_CODE_isMissing": false,
"ORDER_DATE": "1993-11-03:07:51:35",
"ORDER_DATE_isMissing": false,
"PRODUCT_CODE": "PLANE",
"PRODUCT_CODE_isMissing": false,
"ORDER_ID": "600",
"ORDER_ID_isMissing": false,
"PRODUCT_PRICE": null,
"PRODUCT_PRICE_isMissing": true,
"PRODUCT_AMOUNT": null,
"PRODUCT_AMOUNT_isMissing": true,
"TRANSACTION_ID": null,
"TRANSACTION_ID_isMissing": true},
"after": null}

```

10.2.38.4.2.2.3.4 Sample Truncate Message

```

{"table": "GG.TCUSTORD",
"op_type": "T",
"op_ts": "2013-06-02 22:14:41.000000",
"current_ts": "2015-09-18T10:17:49.900000",
"pos": "00000000000000004515",
"primary_keys": ["CUST_CODE", "ORDER_DATE", "PRODUCT_CODE", "ORDER_ID"], "tokens":
{"R": "AADPkVAAEAeqL2AAB"},

```

```
"before": null,
"after": null}
```

10.2.38.4.2.2.4 Avro Schema

Avro schemas are represented as JSONs. Avro schemas define the format of generated Avro messages and are required to serialize and deserialize Avro messages. Avro schemas are generated on a just-in-time basis when the first operation for a table is encountered. Because Avro schemas are specific to a table definition, a separate Avro schema is generated for every table encountered for processed operations. By default, Avro schemas are written to the *GoldenGate_Home/dirdef* directory, although the write location is configurable. Avro schema file names adhere to the following naming convention: *Fully_Qualified_Table_Name.avsc*.

The following is a sample Avro schema for the Avro Operation Format for the samples in the preceding sections:

```
{
  "type" : "record",
  "name" : "TCUSTORD",
  "namespace" : "GG",
  "fields" : [ {
    "name" : "table",
    "type" : "string"
  }, {
    "name" : "op_type",
    "type" : "string"
  }, {
    "name" : "op_ts",
    "type" : "string"
  }, {
    "name" : "current_ts",
    "type" : "string"
  }, {
    "name" : "pos",
    "type" : "string"
  }, {
    "name" : "primary_keys",
    "type" : {
      "type" : "array",
      "items" : "string"
    }
  }, {
    "name" : "tokens",
    "type" : {
      "type" : "map",
      "values" : "string"
    }
  }, {
    "default" : { }
  }, {
    "name" : "before",
    "type" : [ "null", {
      "type" : "record",
      "name" : "columns",
      "fields" : [ {
        "name" : "CUST_CODE",
        "type" : [ "null", "string" ],
        "default" : null
      } ], {
        "name" : "CUST_CODE_isMissing",
        "type" : "boolean"
      } ], {
        "name" : "ORDER_DATE",
        "type" : [ "null", "string" ],
```

```

        "default" : null
    }, {
        "name" : "ORDER_DATE_isMissing",
        "type" : "boolean"
    }, {
        "name" : "PRODUCT_CODE",
        "type" : [ "null", "string" ],
        "default" : null
    }, {
        "name" : "PRODUCT_CODE_isMissing",
        "type" : "boolean"
    }, {
        "name" : "ORDER_ID",
        "type" : [ "null", "string" ],
        "default" : null
    }, {
        "name" : "ORDER_ID_isMissing",
        "type" : "boolean"
    }, {
        "name" : "PRODUCT_PRICE",
        "type" : [ "null", "double" ],
        "default" : null
    }, {
        "name" : "PRODUCT_PRICE_isMissing",
        "type" : "boolean"
    }, {
        "name" : "PRODUCT_AMOUNT",
        "type" : [ "null", "double" ],
        "default" : null
    }, {
        "name" : "PRODUCT_AMOUNT_isMissing",
        "type" : "boolean"
    }, {
        "name" : "TRANSACTION_ID",
        "type" : [ "null", "string" ],
        "default" : null
    }, {
        "name" : "TRANSACTION_ID_isMissing",
        "type" : "boolean"
    }
    ]
} ],
"default" : null
}, {
    "name" : "after",
    "type" : [ "null", "columns" ],
    "default" : null
} ]
}

```

10.2.38.4.2.2.5 Avro Operation Formatter Configuration Properties

Table 10-54 Configuration Properties

Properties	Optional Y/N	Legal Values	Default	Explanation
<code>gg.handler.name.form</code> <code>at.insertOpKey</code>	Optional	Any string	I	Indicator to be inserted into the output record to indicate an insert operation

Table 10-54 (Cont.) Configuration Properties

Properties	Optional Y/N	Legal Values	Default	Explanation
<code>gg.handler.name.form at.updateOpKey</code>	Optional	Any string	U	Indicator to be inserted into the output record to indicate an update operation.
<code>gg.handler.name.form at.deleteOpKey</code>	Optional	Any string	D	Indicator to be inserted into the output record to indicate a delete operation.
<code>gg.handler.name.form at.truncateOpKey</code>	Optional	Any string	T	Indicator to be inserted into the output record to indicate a truncate operation.
<code>gg.handler.name.form at.encoding</code>	Optional	Any legal encoding name or alias supported by Java	UTF-8 (the JSON default)	Controls the output encoding of generated JSON Avro schema. The JSON default is UTF-8. Avro messages are binary and support their own internal representation of encoding.
<code>gg.handler.name.form at.treatAllColumnsAsStrings</code>	Optional	true false	false	Controls the output typing of generated Avro messages. If set to false, then the formatter attempts to map Oracle GoldenGate types to the corresponding Avro type. If set to true, then all data is treated as Strings in the generated Avro messages and schemas.
<code>gg.handler.name.form at.lineDelimiter</code>	Optional	Any string	no value	Inserts delimiter after each Avro message. This is not a best practice, but in certain cases you may want to parse a stream of data and extract individual Avro messages from the stream, use this property to help. Select a unique delimiter that cannot occur in any Avro message. This property supports CDATA[] wrapping.
<code>gg.handler.name.form at.schemaDirectory</code>	Optional	Any legal, existing file system path.	./dirdef	The output location of generated Avro schemas.
<code>gg.handler.name.form at.wrapMessageInGenericAvroMessage</code>	Optional	true false	false	Wraps Avro messages for operations from the source trail file in a generic Avro wrapper message. For more information, see Generic Wrapper Functionality .
<code>gg.handler.name.form at.iso8601Format</code>	Optional	true false	true	The format of the current timestamp. By default the ISO 8601 is set to false, removes the T between the date and time in the current timestamp, which outputs a space instead.

Table 10-54 (Cont.) Configuration Properties

Properties	Optional Y/N	Legal Values	Default	Explanation
<code>gg.handler.name.format.includeIsMissingFields</code>	Optional	true false	false	Set to true to include a <code>{column_name}_isMissing</code> boolean field for each source field. This field allows downstream applications to differentiate if a null value is null in the source trail file (value is false) or is missing in the the source trail file (value is true).
<code>gg.handler.name.format.oracleNumberScale</code>	Optional	Any integer value from 0 to 38.	None	Allows you to set the scale on the Avro decimal data type. Only applicable when you set <code>enableDecimalLogicalType=true</code> . The Oracle NUMBER is a proprietary numeric data type of Oracle Database that supports variable precision and scale. Precision and scale are variable on a per instance of the Oracle NUMBER data type. Precision and scale are required parameters when generating the Avro decimal logical type. This makes mapping of Oracle NUMBER data types into Avro difficult because there is no way to deterministically know the precision and scale of an Oracle NUMBER data type when the Avro schema is generated. The best alternative is to generate a large Avro decimal data type a precision of 164 and a scale of 38, which should hold any legal instance of Oracle NUMBER. While this solves the problem of precision loss when converting Oracle Number data types to Avro decimal data types, you may not like that Avro decimal data types when retrieved from Avro messages downstream have 38 digits trailing the decimal point.

Table 10-54 (Cont.) Configuration Properties

Properties	Optional Y/N	Legal Values	Default	Explanation
<code>gg.handler.name.format.mapOracleNumbersAsStrings</code>	Optional	true false	false	This property is only applicable if decimal logical types are enabled via the property <code>gg.handler.name.format.enableDecimalLogicalType=true</code> . Oracle numbers are especially problematic because they have a large precision (168) and floating scale of up to 38. Some analytical tools, such as Spark cannot read numbers that large. This property allows you to map those Oracle numbers as strings while still mapping the smaller numbers as decimal logical types.
<code>gg.handler.name.format.enableTimestampLogicalType</code>	Optional	true false	false	Set to true to map source date and time data types into the Avro TimestampMicros logical data type. If <code>gg.handler.name.format.enableTimestampLogicalType</code> is set to true and <code>gg.format.timestamp</code> is unset, then replicat will abend with a configuration exception. The variable <code>gg.format.timestamp</code> must be configured to provide a mask for the source date and time data types to make sense of them. The Avro TimestampMicros is part of the Avro 1.8 specification.
<code>gg.handler.name.format.enableDecimalLogicalType</code>	Optional	true false	false	Enables the use of Avro decimal logical types. The decimal logical type represents numbers as a byte array and can provide support for much larger numbers than can fit in the classic 64-bit long or double data types.
<code>gg.handler.name.format.mapLargeNumbersAsStrings</code>	Optional	true false	false	Oracle GoldenGate supports the floating point and integer source datatypes. Some of these datatypes may not fit into the Avro primitive double or long datatypes. Set this property to true to map the fields that do not fit into the Avro primitive double or long datatypes to Avro string.

Table 10-54 (Cont.) Configuration Properties

Properties	Optional Y/N	Legal Values	Default	Explanation
<code>gg.handler.name.form</code> <code>at.metaColumnsTemplate</code>	Optional	See Metacolumn Keywords	None	The current meta column information can be configured in a simple manner and removes the explicit need to use: <pre>insertOpKey updateOpKey deleteOpKey truncateOpKey includeTableName includeOpTimestamp includeOpType includePosition includeCurrentTimestamp, useIso8601Format</pre> It is a comma-delimited string consisting of one or more templated values that represent the template. For more information about the Metacolumn keywords, see Metacolumn Keywords.

Table 10-54 (Cont.) Configuration Properties

Properties	Optional Y/N	Legal Values	Default	Explanation
<code>gg.handler.name.form at.maxPrecision</code>	Optional	None	Positive Integer	Allows you to set the maximum precision for Avro decimal logical types. Consuming applications may have limitations on Avro precision (that is, Apache Spark supports a maximum precision of 38).

Warning

Configuration of this property is not without risk.

The NUMBER type in an Oracle RDBMS supports a maximum precision of 164. Configuration of this property likely means you are casting larger source numeric types to smaller target numeric types. If the precision of the source value is greater than the configured precision, a runtime exception occurs and the replicat process will abend. That behavior is not a bug. That is the expected behavior.

10.2.38.4.2.2.6 Review a Sample Configuration

The following is a sample configuration for the Avro Operation Formatter in the Java Adapter property file:

```
gg.handler.hdfs.format=avro_op
gg.handler.hdfs.format.insertOpKey=I
gg.handler.hdfs.format.updateOpKey=U
gg.handler.hdfs.format.deleteOpKey=D
gg.handler.hdfs.format.truncateOpKey=T
gg.handler.hdfs.format.encoding=UTF-8
gg.handler.hdfs.format.wrapMessageInGenericAvroMessage=false
```

10.2.38.4.2.2.7 Metadata Change Events

If the replicated database and upstream Oracle GoldenGate replication process can propagate metadata change events, the Avro Operation Formatter can take action when metadata changes. Because Avro messages depend closely on their corresponding schema, metadata changes are important when you use Avro formatting.

An updated Avro schema is generated as soon as a table operation occurs after a metadata change event.

You must understand the impact of a metadata change event and change downstream targets to the new Avro schema. The tight dependency of Avro messages to Avro schemas may result in compatibility issues. Avro messages generated before the schema change may not be able to be deserialized with the newly generated Avro schema. Conversely, Avro messages generated after the schema change may not be able to be deserialized with the previous Avro schema. It is a best practice to use the same version of the Avro schema that was used to generate the message.

For more information, consult the Apache Avro documentation.

10.2.38.4.2.2.8 Special Considerations

This section describes these special considerations:

- [Troubleshooting](#)
- [Primary Key Updates](#)
- [Generic Wrapper Message](#)

10.2.38.4.2.2.8.1 Troubleshooting

Because Avro is a binary format, it is not human readable. However, when the `log4j` Java logging level is set to `TRACE`, Avro messages are deserialized and displayed in the log file as a JSON object, letting you view the structure and contents of the created Avro messages. Do not enable `TRACE` in a production environment, as it has a substantial impact on performance.

10.2.38.4.2.2.8.2 Primary Key Updates

The Avro Operation Formatter creates messages with complete data of before-image and after-images for update operations. Therefore, the Avro Operation Formatter requires no special treatment for primary key updates.

10.2.38.4.2.2.8.3 Generic Wrapper Message

Because Avro messages are not self describing, the receiver of the message must know the schema associated with the message before the message can be deserialized. Avro messages are binary and provide no consistent or reliable way to inspect the message contents in order to ascertain the message type. Therefore, Avro can be troublesome when messages are interlaced into a single stream of data such as Kafka.

The Avro formatter provides a special feature to wrap the Avro message in a generic Avro message. You can enable this functionality by setting the following configuration property:

```
gg.handler.name.format.wrapMessageInGenericAvroMessage=true
```

The generic message is Avro message wrapping the Avro payload message that is common to all Avro messages that are output. The schema for the generic message is `generic_wrapper.avsc` and is written to the output schema directory. This message has the following three fields:

- `table_name`: The fully qualified source table name.

- `schema_fingerprint` : The fingerprint of the of the Avro schema generating the messages. The fingerprint is generated using the `parsingFingerprint64(Schema s)` method on the `org.apache.avro.SchemaNormalization` class.
- `payload`: The wrapped Avro message.

The following is the Avro Formatter generic wrapper schema:

```
{
  "type" : "record",
  "name" : "generic_wrapper",
  "namespace" : "oracle.goldengate",
  "fields" : [ {
    "name" : "table_name",
    "type" : "string"
  }, {
    "name" : "schema_fingerprint",
    "type" : "long"
  }, {
    "name" : "payload",
    "type" : "bytes"
  } ]
}
```

10.2.38.4.2.3 Avro Object Container File Formatter

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) can write to HDFS in Avro Object Container File (OCF) format. Avro OCF handles schema evolution more efficiently than other formats. The Avro OCF Formatter also supports compression and decompression to allow more efficient use of disk space.

The HDFS Handler integrates with the Avro formatters to write files to HDFS in Avro OCF format. The Avro OCF format is required for Hive to read Avro data in HDFS. The Avro OCF format is detailed in the Avro specification, see <http://avro.apache.org/docs/current/spec.html#Object+Container+Files>.

You can configure the HDFS Handler to stream data in Avro OCF format, generate table definitions in Hive, and update table definitions in Hive in the case of a metadata change event.

- [Avro OCF Formatter Configuration Properties](#)

10.2.38.4.2.3.1 Avro OCF Formatter Configuration Properties

Properties	Optional / Required	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.format.insertOpKey</code>	Optional	Any string	I	Indicator to be inserted into the output record to indicate an insert operation.
<code>gg.handler.name</code> <code>.format.updateOpKey</code>	Optional	Any string	U	Indicator to be inserted into the output record to indicate an update operation.
<code>gg.handler.name</code> <code>.format.truncateOpKey</code>	Optional	Any string	T	Indicator to be truncated into the output record to indicate a truncate operation.

Properties	Optional / Required	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.format.deleteOpKey</code>	Optional	Any string	D	Indicator to be inserted into the output record to indicate a truncate operation.
<code>gg.handler.name</code> <code>.format.encoding</code>	Optional	Any legal encoding name or alias supported by Java.	UTF-8	Controls the output encoding of generated JSON Avro schema. The JSON default is UTF-8. Avro messages are binary and support their own internal representation of encoding.
<code>gg.handler.name</code> <code>.format.treatAllColumnsAsStrings</code>	Optional	true false	false	Controls the output typing of generated Avro messages. When the setting is false, the formatter attempts to map Oracle GoldenGate types to the corresponding Avro type. When the setting is true, all data is treated as strings in the generated Avro messages and schemas.

Properties	Optional / Required	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.format.pkUpdateHandling</code>	Optional	<code>abend</code> <code>update</code> <code>delete-insert</code>	<code>abend</code>	Controls how the formatter should handle update operations that change a primary key. Primary key operations can be problematic for the Avro Row formatter and require special consideration by you. <code>abend</code> : the process will terminate. <code>update</code> : the process handles this as a normal update <code>delete and insert</code>: the process handles this operation as a delete and an insert. The full before image is required for this feature to work properly. This can be achieved by using full supplemental logging in Oracle. Without full before and after row images the insert data will be incomplete.
<code>gg.handler.name</code> <code>.format.generateSchema</code>	Optional	<code>true</code> <code>false</code>	<code>true</code>	Because schemas must be generated for Avro serialization to <code>false</code> to suppress the writing of the generated schemas to the local file system.

Properties	Optional / Required	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.format.schemaDirectory</code>	Optional	Any legal, existing file system path	<code>./dirdef</code>	The directory where generated Avro schemas are saved to the local file system. This property does not control where the Avro schema is written to in HDFS; that is controlled by an HDFS Handler property.
<code>gg.handler.name</code> <code>.format.iso8601Format</code>	Optional	<code>true false</code>	<code>true</code>	By default, the value of this property is <code>true</code> , and the format for the current timestamp is ISO8601. Set to <code>false</code> to remove the T between the date and time in the current timestamp and output a space instead.
<code>gg.handler.name</code> <code>.format.versionSchemas</code>	Optional	<code>true false</code>	<code>false</code>	If set to <code>true</code> , an Avro schema is created in the schema directory and versioned by a time stamp. The schema uses the following format: <i>fully_qualified_table_name_time_stamp.avsc</i>

10.2.38.4.3 Cloud Event Formatter

- [Using the Cloud Event Formatter](#)
- [Operation Data Formatting Details](#)
- [Row Data Formatting Details](#)
- [Updates to Schema Attributes for the Cloud Event Formatter](#)
- [Sample Configuration](#)
- [Cloud Event Formatter Configuration Properties](#)

10.2.38.4.3.1 Using the Cloud Event Formatter

CloudEvents is a specification for describing event data in common formats to provide interoperability across services, platforms and systems. CloudEvents format defines the list of

attributes to describe the event, essentially an envelope with a set of mandatory and optional attributes.

Currently, CloudEvents format is limited to the JSON event format.

When CloudEvents format is enabled in the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) targets, the final JSON records will look like as follows where data field contains the original data records.

For Insert:

```
{ "specversion": "1.0", "id": "OGG-000000000000000004120", "source": "uri:slcaa318:oldh
ome:qastaf:ggs_home:yuga:ora:v122_rc3:ETEST", "type": "QASOURCE.TCUSTORD", "datacont
enttype": "application/json", "data":
{ "table": "QASOURCE.TCUSTORD", "op_type": "I", "op_ts": "2015-11-05
18:45:39.000000", "current_ts": "2024-07-30
05:52:26.030000", "pos": "000000000000000004120", "after":
{ "CUST_CODE": "DAVE", "ORDER_DATE": "1993-11-03
07:51:35", "PRODUCT_CODE": "PLANE", "ORDER_ID": 600, "PRODUCT_PRICE": 135000.00, "PRODUC
T_AMOUNT": 2, "TRANSACTION_ID": 200}}}
```

For Updates:

```
{ "specversion": "1.0", "id": "OGG-000000000000000005100", "source": "uri:slcaa318:oldh
ome:qastaf:ggs_home:yuga:ora:v122_rc3:ETEST", "type": "com.oracle.goldengate", "data
contenttype": "application/json", "data": { "before": { "CUST_CODE": "ANN", "NAME": "ANN'S
BOATS", "CITY": "SEATTLE", "STATE": "WA"}, "after": { "CUST_CODE": "ANN", "CITY": "NEW
YORK", "STATE": "NY"}}}
```

For Deletes:

```
{ "specversion": "1.0", "id": "OGG-000000000000000005272", "source": "uri:slcaa318:oldh
ome:qastaf:ggs_home:yuga:ora:v122_rc3:ETEST", "type": "QASOURCE.TCUSTORD", "datacont
enttype": "application/json", "data":
{ "table": "QASOURCE.TCUSTORD", "op_type": "D", "op_ts": "2015-11-05
18:45:39.000000", "current_ts": "2024-07-30
05:52:26.052000", "pos": "000000000000000005272", "before":
{ "CUST_CODE": "DAVE", "ORDER_DATE": "1993-11-03
07:51:35", "PRODUCT_CODE": "PLANE", "ORDER_ID": 600, "PRODUCT_PRICE": 135000.00, "PRODUC
T_AMOUNT": 2, "TRANSACTION_ID": 200}}}
```

CloudEvents formatter mandates the following four attributes for each event record.

"id", "source", "specversion", "type"

specversion

This is the non-empty Cloud Events specification version string, and it is "1.0" in the current release.

For example: "specversion": "1.0"

id

This is the unique identifier for the event records. It is defined as a non-empty string. By default, the position of the record prefixed with "OGG-" is used for event record ID.

For example: "id": "OGG-000000000000000005100"

This can be overridden by using the property `idmappingtemplate`.

For example: `gg.handler.kafkahandler.format.idmappingtemplate=OGG-#{position}`

type

This is the type of the event related to the originating occurrence. Defaulting to `com.oracle.goldengate` for GG for DAA and can be overridden by using the property `typemappingtemplate`.

For example: `gg.handler.kafkahandler.format.typemappingtemplate=#{tablename}`

source

The source field is the context in which the occurrence happened, and it is in the format of URI-Reference. The source trail producer obtained from the source trail header file is used as this field value.

For example:

```
"source": "uri:slcaa318::oldhome:qastaf:ggs_home:yuga:ora:v122_rc3:ETEST"
```

This can be overridden by using the property `sourcemappingtemplate`.

For example: `gg.handler.kafkahandler.format.sourcemappingtemplate=#{tablename}`

data

The data field contains the original event data. It can only be in JSON object type. Defaulted to represent the data it in Operation Data format.

Additional attributes:

Additional attributes can be sent upon the configuration of the specified explicit properties.

Following are the additional attributes.

Datacontenttype:

The valid value for now is "application/json".

Subject:

Subject of the event in the context of the event producer. Can be configured using `subjectMappingTemplate` property.

For example: `gg.handler.kafkahandler.format.subjectMappingTemplate=OGGDAA-KafkaStreams`

Time

Timestamp of when the occurrence happened. Can be controlled by template keywords.

For example: `TimeMappingTemplategg.handler.kafkahandler.format.timeMappingTemplate=#{opTimestamp}`

10.2.38.4.3.2 Operation Data Formatting Details

The Cloud Event formatter represents the event data in Operation Data Formatting where the data is represented by before and after members that are objects. These objects contain members whose keys are the column names and whose values are the column values.

Operation data is modeled as follows:

- **Inserts:** Includes the after-image data.
- **Updates:** Includes both the before-image and the after-image data.
- **Deletes:** Includes the before-image data.

For example:

```
{ "specversion": "1.0", "id": "OGG-000000000000000005100", "source": "uri:slcaa318::oldh
ome:qastaf:ggs_home:yuga:ora:v122_rc3:ETEST", "type": "com.oracle.goldengate", "data
contenttype": "application/json", "data": { "before": { "CUST_CODE": "ANN", "NAME": "ANN'S
BOATS", "CITY": "SEATTLE", "STATE": "WA"}, "after": { "CUST_CODE": "ANN", "CITY": "NEW
YORK", "STATE": "NY"} } }
```

10.2.38.4.3.3 Row Data Formatting Details

When configured Row Data Formatting, the data attribute would represent the event data as objects. These objects contain members whose keys are the column names and whose values are the column values.

Row data is modeled as follows:

- **Inserts:** Includes the after-image data.
- **Updates:** Includes the after-image data.
- **Deletes:** Includes the before-image data.

For example:

```
{ "specversion": "1.0", "id": "OGG-000000000000000005100", "source": "uri:slcaa318::oldh
ome:qastaf:ggs_home:yuga:ora:v122_rc3:ETEST", "type": "com.oracle.goldengate", "data
contenttype": "application/json", "data": { "CUST_CODE": "ANN", "CITY": "NEW
YORK", "STATE": "NY"} }
```

10.2.38.4.3.4 Updates to Schema Attributes for the Cloud Event Formatter

An Avro schema for a designated table is published the first time an operation pertaining to that table is detected. Within the generated schema, the title and id attributes play a crucial role in identifying its connection to the Cloud Event formatter. The title attribute denotes the specific table, whereas the id attribute features the fully qualified table name, prefixed with `ogg:dataStream:DMLRecord:`.

Sample Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Oracle GoldenGate JSON Schema for DML Record from QASOURCE.TCUSTMER",
  "description": "JSON schema for table QASOURCE.TCUSTMER",
  "id": "ogg:dataStream:DMLRecord:QASOURCE.TCUSTMER",
  "definitions": {
    "row": {
      "type": "object",
      "properties": {
        "CUST_CODE": {
```

```

        "type": [
            "string",
            "null"
        ]
    },
    "NAME": {
        "type": [
            "string",
            "null"
        ]
    },
    "CITY": {
        "type": [
            "string",
            "null"
        ]
    },
    "STATE": {
        "type": [
            "string",
            "null"
        ]
    }
},
"additionalProperties": false
},
"tokens": {
    "type": "object",
    "description": "Token keys and values are free form key value pairs.",
    "properties": {
    },
    "additionalProperties": true
}
},
"type": "object",
"properties": {
    "table": {
        "description": "The MetaColumn table",
        "type": "string"
    },
    "op_type": {
        "description": "The MetaColumn op_type",
        "type": "string"
    },
    "op_ts": {
        "description": "The MetaColumn op_ts",
        "type": "string"
    },
    "current_ts": {
        "description": "The MetaColumn current_ts",
        "type": "string"
    },
    "pos": {
        "description": "The MetaColumn pos",
        "type": "string"
    },
    "before": {
        "$ref": "#/definitions/row"
    },
    "after": {
        "$ref": "#/definitions/row"
    }
}

```

```

    },
    "required": [
        "table",
        "op_type",
        "op_ts",
        "current_ts",
        "pos"
    ],
    "additionalProperties": false
}

```

10.2.38.4.3.5 Sample Configuration

```

gg.handler.kafkahandler.format=json_cloudevent

gg.handler.kafkahandler.format.metaColumnsTemplate=${objectname[table]},${
{optype[op_type]},${timestamp[op_ts]},${currenttimestamp[current_ts]},${position[pos]}

gg.handler.kafkahandler.format.idmappingtemplate=OGG-${position}
gg.handler.kafkahandler.format.typemappingtemplate=${fullyQualifiedTableName}
gg.handler.kafkahandler.format.sourcemappingtemplate=${fullyQualifiedTableName}
gg.handler.kafkahandler.format.subjectmappingtemplate=From OGGDAA
gg.handler.kafkahandler.format.timemappingtemplate=${opTimestamp}
gg.handler.kafkahandler.format.dataContentType=application/json
gg.handler.kafkahandler.format.specversion=V1

```

10.2.38.4.3.6 Cloud Event Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler.name .format	Optional	json_cloudevent json_op_cloudev ent json_row_cloude vent	None	Controls whether the generated Cloud Event's JSON output messages are operation modeled or row modeled. Set to json for operation modeled or json_row for row modeled.
gg.handler.name .format.idmappi ngtemplate	Optional	See Template Keywords	OGG-\${position}	This is the unique identifier for the event records. It is defined as a non-empty string. By default, the position of the record prefixed with OGG- is used for event record ID. This can be overridden by using template keywords.

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler.name .format.typemap pingtemplate	Optional	See Template Keywords	com.oracle.gold engate	This is the type of the event related to the originating occurrence. Defaulting to com.oracle.gold engate for Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) and can be overridden by using Template Keywords .
gg.handler.name .format.sourcem appingtemplate	optional	See Template Keywords	Trail header file value.	The "source" field is the context in which the occurrence happened, and it is in the format of URI-Reference. The source trail producer obtained from the source trail header file is used as this field value. can be overridden by using Template Keywords .
gg.handler.name .format.subject mappingtemplate	Optional	See Template Keywords	None	Subject of the event in the context of the event producer. Can be configured using using Template Keywords .
gg.handler.name .format.timemap pingtemplate	Optional	See Template Keywords	None	Timestamp of when the occurrence happened. Can be controlled by Template Keywords .
gg.handler.name .format.dataCon tentType	Optional	application/ json	application/ json	ContentType of the data.
gg.handler.name .format.specver sion	Optional	Valid SpecVersion supported by CloudEvents.	V1	Cloud Events specification version.

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler.name .format.metaColumnsTemplate	Optional	See Metacolumn Keywords .	None	The current meta column information can be configured in a simple manner and removes the explicit need to use: <pre>insertOpKey updateOpKey deleteOpKey truncateOpKey includeTableName includeOpTimestamp includeOpType includePosition includeCurrentTimestamp, useIso8601Format</pre> It is a comma-delimited string consisting of one or more templated values that represent the template. For more information about the Metacolumn keywords, see Metacolumn Keywords. This is an example that would produce a list of metacolumns: <pre>\$ {optype}, \$ {token.ROWID} , \$ {sys.username },\$ {currenttimestamp}</pre>

10.2.38.4.4 Existing Avro Formatter

The existing Avro formatter allows users to map trail records into their existing Avro schemas.

- [Prerequisites](#)

- [Supported Output Formats](#)
- [Support for Mapping to Nested Avro Fields](#)
- [Support for Mapping to Nested Avro Fields](#)
- [Configuration](#)
- [Unsupported Features](#)

10.2.38.4.4.1 Prerequisites

To enable this feature, the Avro Metadata Provider must be used. Ensure to set the following configuration parameters:

```
gg.mdp.type=avro
gg.mdp.schemaFilePath=
```

10.2.38.4.4.2 Supported Output Formats

The Existing Avro Schema formatter can generate messages in following formats:

- Raw avro – Generally used with Kafka or other integrations when sending individual messages.
- Avro OCF (object container file) – Generally used when generating files such as with the File Writer Handler. This is the format which has the Avro schema in the file header and then contains one or more records which conform to the schema.
- JSON – It is possible to convert the Avro Generic Record to JSON and publish the JSON. This can be used for debugging since JSON provides human readability whereas Avro does not. Additionally, it could be used if you wish to send data in JSON formats which support nested hierarchical structures.

10.2.38.4.4.3 Support for Mapping to Nested Avro Fields

The Existing Avro Schema Formatter supports mapping into Avro hierarchical structures. This is one of the powerful features of the Existing Avro Schema formatter. As previously stated, the GoldenGate trail structure is a flat structure of key-value pairs which model the flat column name and column value structure of an RDBMS. Avro can be a hierarchical structure where records can be nested inside of fields and records. The Avro Metadata provider has been modified to traverse the Avro schema and to flatten the Avro structure to return the target metadata to the GoldenGate mapping.

The following is an example of the hierarchical Avro schema:

```
{
  "type" : "record",
  "name" : "TCUSTOMERNEST",
  "namespace" : "QASOURCE",
  "fields" : [ {
    "name" : "CUST_CODE",
    "type" : [ "null", "string" ],
    "default" : null,
    "primary_key": true
  }, {
    "name" : "SUB",
    "type" : {
      "type" : "record",
      "name" : "subrecord",
      "fields" : [ {
        "name" : "NAME",
```

```

        "type" : [ "null", "string" ],
        "default" : null
    }, {
        "name" : "STATE",
        "type" : [ "null", "string" ],
        "default" : null
    }, {
        "name" : "CITY",
        "type" : [ "null", "string" ],
        "default" : null
    } ]
    }
} ]
}

```

Field CUST_CODE is at the root record level. However, fields NAME, STATE, and CITY are fields nesting inside of field SUB which is at the root record level. The meta provider must flatten the Avro structure and return it to GoldenGate so that values can be mapped into the Avro hierarchical structure. The "-" (dash) character is used to designate or delimit levels of the hierarchical structure. Avro does not support the "-" character in field names. Avro field names only support upper case A to Z, lower case a to z, numbers, and the "_" (underbar) character. Therefore the "-" does not conflict with Avro name. The Avro metadata provider returns the following fields for the above schema.

```

CUST_CODE
SUB-NAME
SUB-STATE
SUB-CITY

```

You can then map data into these fields using the replicat mapping statement. For example:

```

MAP QASOURCE.TCUSTMER, TARGET QASOURCE.TCUSTMERNEST COLMAP
(CUST_CODE=CUST_CODE, SUB-NAME=NAME, SUB-CITY=CITY, SUB-STATE=STATE);

```

Avro has no restriction how deep fields can be nested. It can be confusing to customers who are looking at an Avro schema to understand what the flattened structure looks like. To help the flattened structure is logged to the <replicat name>.log file. Therefore, if you run the replicat, then you can examine the log file to see the flattened fields.

```

INFO 2023-08-09 14:54:42.000242 [main] -
Retrieved Avro metadata for table [QASOURCE.TCUSTMERNEST]
Field name [CUST_CODE] Type [STRING] Nullable [true].
Field name [SUB-NAME] Type [STRING] Nullable [true].
Field name [SUB-STATE] Type [STRING] Nullable [true].
Field name [SUB-CITY] Type [STRING] Nullable [true].
End retrieved metadata.

```

10.2.38.4.4.4 Support for Mapping to Nested Avro Fields

The GoldenGate trail file supports the following three states for column data.

- Column has a value.
- Column value is null.
- Column value is missing.

If the source field is null the order of operations for target value resolution is the following:

- If the mapped Avro field is nullable, then null will be mapped to the target field.

- If the mapped Avro field is not nullable, then the default value for the field will be mapped as defined in the Avro schema.
- If the target field is not nullable and provides no default value, then this is an exception. The replicat will log an exception and abend.

If the source field is missing the order of operations for target value resolution is the following:

- If a default value is defined in the Avro schema then the default value will be mapped into the target field.
- If no default value is defined in the Avro schema, and the field is nullable, then null will be mapped into the target field.
- If the target field is not nullable and provides no default value, then this is an exception. The replicat will log an exception and abend.

Additionally, it is important to understand that the default value for a field may be null.

Examples of Avro field definitions:

String field not nullable and no default.

```
{"name": "STRING_REQUIRED", "type": "string"}
```

String field not nullable with default value.

```
{"name": "STRING_REQUIRED", "type": "string", "default": "mydefault"}
```

String field nullable without default.

```
{"name": "STRING_NULLABLE", "type": ["null", "string"]}
```

String field nullable with null default.

```
{"name": "STRING_NULLABLE", "type": ["null", "string"], "default": null}
```

String field nullable with non-null default.

```
{"name": "STRING_NULLABLE", "type": ["string", "null"], "default": "mydefault"}
```

Note, it is an Avro requirement that default value apply to the first type in the type union. None of the of the following are legal. These will fail when the Avro library parses the schema and replicat will abend.

```
{"name": "STRING_NULLABLE", "type": ["string", "null"], "default": null}
{"name": "STRING_NULLABLE", "type": ["null", "string"], "default": "mydefault"}
{"name": "STRING_REQUIRED", "type": "string", "default": null}
```

10.2.38.4.4.5 Configuration

Table 10-55 Configuration Properties

Configuration Properties	Optional/Required	Legal Values	Default	Explanation
<code>gg.handler.name.format</code>	true	<code>existing_avro_schema</code> <code>existing_avro_schema_ocf</code> <code>existing_avro_schema_json</code>	None	
<code>gg.handler.name.format.fixedPaddingByte</code>	false	A byte in hex format. For example an ASCII space would be represented as 20.	00 (Hex value 00 often referred to as the null byte)	Only applicable if source schemas contain fields defined as fixed. Avro fixed type is a fixed length binary record. If the data is not sufficient to fill the space, then the field must be padded with bytes. This allows the user to configure that padding byte. So for example, <code>gg.handler.name.format.fixedPaddingByte=20</code> to set to the space character.
<code>gg.handler.name.format.encoding</code>	false	A Java legal encoding like UTF-8.	The system default system encoding.	Only applicable if <code>gg.handler.name.format=existing_avro_schema_json</code> . Used to set the encoding of generated JSON data.
<code>gg.handler.name.format.localZoneId</code>	false	A Java legal time zone.	The system default time zone.	This is to support Avro logical types local-timestamp-millis and local-timestamp-micros. The object from OGG is a Java Instant which is in UTC. It must be converted to a LocalDateTime object which requires a time zone to convert.

Table 10-55 (Cont.) Configuration Properties

Configuration Properties	Optional/Required	Legal Values	Default	Explanation
gg.handler.name.format.jsonDelimiter	false	String	None	Only applicable if gg.handler.name.format=existing_avro_schema_json. Allows a delimiter to be inserted between generated JSON documents. The most common use case is to insert a line feed character between the JSON documents. For example: gg.handler.name.format=CDATA[\n] CDATA[] is supported to preserve whitespace in the configuration.

10.2.38.4.4.6 Unsupported Features

Maps and Arrays

Neither the Avro Metadata Provider nor the Existing Avro schema formatter support Avro arrays or maps. Avro arrays and maps are containers, which can hold 0 to N elements. The Oracle GoldenGate trail structure is a flat structure of key and value pairs which in turn models the flat column name and column value structure of an RDBMS. Target metadata must be made to be a flat structure and the number of elements must be deterministic. Arrays and maps are problematic because they can contain 0 to N elements. Different messages can contain different numbers of elements for the collections.

Recursive Record Definitions

Avro supports recursive record definitions. A recursive record definition is where an element in record A contains a field which references record B. The definition of record B contains a field which references record A. The following is an example of a recursive record definition.

```
{
  "type" : "record",
  "name" : "TCUSTOMERINVALID",
  "namespace" : "QASOURCE",
  "fields" : [ {
    "name" : "CUST_CODE",
    "type" : [ "null", "string" ],
    "default" : null,
    "primary_key": true
  }, {
    "name" : "SUB1",
    "type" : [ "null", {
      "type" : "record",
```

```

    "name" : "subrecord",
    "fields" : [ {
      "name" : "NAME",
      "type" : [ "null", "string" ],
      "default" : null
    }, {
      "name" : "SUB2",
      "type" : {
        "type" : "record",
        "name" : "subrecord2",
        "fields" : [ {
          "name" : "RECURSIVEELEMENT",
          "type" : [ "null", "subrecord" ],
          "default" : null
        } ]
      }
    } ]
  }, {
    "name" : "STATE",
    "type" : [ "null", "string" ],
    "default" : null
  } ]
}

```

Element SUB1 is a record of type subrecord. This record contains field SUB2 which is of type subrecord2. Type subrecord 2 contains a field RECURSIVEELEMENT, which is of type subrecord. While this is a legal Avro schema it is problematic for the Avro Metadata Provider to process. The structure is recursive and therefore, the schema definition is by default infinitely deep. The Avro Metadata Provider must flatten the Avro schema into a deterministic flat structure of element names. The recursive nature of the schema definition prevents the Avro metadata provider from being able to generate a deterministic flat structure of element names. Parsing of Avro schemas with a recursive definition results in the following run time exception:

```

ERROR 2023-08-09 15:16:19.000560 [main] - The Avro schema for table
[QASOURCE.TCUSTMERINVALID] encountered a recursive reference to a record schema. The
recursively referenced record schema is namespace [QASOURCE] name [subrecord2]. Schemas
with recursively referenced record schemas are not supported.
ERROR 2023-08-09 15:16:19.000563 [main] - AVRO-00005 Unable to retrieve table metadata
table : [QASOURCE.TCUSTMERINVALID].
oracle.goldengate.datasource.metadata.provider.ResolutionException: The Avro schema for
table [QASOURCE.TCUSTMERINVALID] encountered a recursive reference to a record schema.
The recursively referenced record schema is namespace [QASOURCE] name [subrecord2].
Schemas with recursively referenced record schemas are not supported.

```

Non-discrete Unions

The Avro MDP only supports Unions of null and another type. For example: "type" : ["null", "string"], The following is an Avro union type of double and string. This is legal in Avro; however, it cannot be supported in the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) metadata provider. The problem is that the field needs to be assigned to a discrete type. This field type, by definition, is not discrete. It can be either a double or a string: "type" : ["double", "string"],

Metacolumns Configuration

Metacolumn configuration is not supported with the existing Avro schema formatter. The following configuration will cause the replicat process to abend:

```

gg.handler.name.format.metaColumnsTemplate=

```

The reason is that the input data from replicat is being mapped into an existing Avro schema. The result of configuring metacolumns is to add additional fields to the output data which is contrary to the idea of outputting the data which conforms to the provided Avro schema. It is possible to output operation metadata into the output records. To accomplish this the user should modify the Avro schema to add one or more fields to hold the desired operation metadata. Then the replicat mapping statement can be used to map operation metadata into the fields.

Sample Configuration

The following is example configuration of end to end configuration using the existing Avro schema formatter functionality.

Replicat .prm file: mdp.prm

```
REPLICAT mdp
-- Trail file for this example is located in "AdapterExamples/trail" directory
-- Command to add REPLICAT
-- add replicat mdp, exttrail AdapterExamples/trail/tr
TARGETDB LIBFILE libggjava.so SET property=dirprm/mdp.props
REPORTCOUNT EVERY 1 MINUTES, RATE
GROUPTRANSOPS 1000
MAP QASOURCE.TCUSTMER, TARGET QASOURCE.TCUSTMERNEST COLMAP (SUB-NAME=NAME, SUB-
CITY=CITY, SUB-STATE=STATE, CUST_CODE=CUST_CODE);
```

Replicat Properties File: mdp.props

```
gg.handlerlist=filewriter

#The File Writer Handler
gg.handler.filewriter.type=filewriter
gg.handler.filewriter.mode=op
gg.handler.filewriter.pathMappingTemplate=./dirout
gg.handler.filewriter.stateFileDirectory=./dirsta
gg.handler.filewriter.fileNameMappingTemplate=${fullyQualifiedTableName}_${
currentTimestamp}.txt
gg.handler.filewriter.fileRollInterval=7m
gg.handler.filewriter.finalizeAction=none
gg.handler.filewriter.inactivityRollInterval=7m
gg.handler.filewriter.format=existing_avro_schema_ocf
gg.handler.filewriter.includetokens=true
gg.handler.filewriter.partitionByTable=true
gg.handler.filewriter.rollOnShutdown=true

gg.format.timestamp=yyyy-MM-dd HH:mm:ss.SSS

#Avro Metadata provider must be enabled.
gg.mdp.type=avro
gg.mdp.schemaFilePath=/scratch/tbcampbe/ggwork/gg23.1.0.0.WORKING5/avromdp
```

Example Avro Schema

```
{
  "type" : "record",
  "name" : "TCUSTMERNEST",
  "namespace" : "QASOURCE",
  "fields" : [ {
    "name" : "CUST_CODE",
    "type" : [ "null", "string" ],
    "default" : null,
    "primary_key": true
  }, {
    "name" : "SUB",
```

```

    "type" : {
      "type" : "record",
      "name" : "subrecord",
      "namespace" : "oracle.goldengate",
      "fields" : [ {
        "name" : "NAME",
        "type" : [ "null", "string" ],
        "default" : null
      }, {
        "name" : "STATE",
        "type" : [ "null", "string" ],
        "default" : null
      }, {
        "name" : "CITY",
        "type" : [ "null", "string" ],
        "default" : null
      } ]
    }
  }
}

```

10.2.38.4.5 Using the Delimited Text Formatter

The Delimited Text Formatter formats database operations from the source trail file into a delimited text output. Each insert, update, delete, or truncate operation from the source trail is formatted into an individual delimited message. Delimited text output includes a fixed number of fields for each table separated by a field delimiter and terminated by a line delimiter. The fields are positionally relevant. Many Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) analytical tools including Hive work well with HDFS files that contain delimited text. Column values for an operation from the source trail file can have one of three states: the column has a value, the column value is null, or the column value is missing. By default, the delimited text maps these column value states into the delimited text output as follows:

- Column has a value: The column value is output.
- Column value is null: The default output value is NULL. The output for the case of a null column value is configurable.
- Column value is missing: The default output value is an empty string (""). The output for the case of a missing column value is configurable.
- [Using the Delimited Text Row Formatter](#)
The Delimited Text Row Formatter is the Delimited Text Formatter that was included a release prior to the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 19.1.0.0.0 release. It writes the after change data for inserts and updates, and before change data for deletes.
- [Delimited Text Operation Formatter](#)
The Delimited Text Operation Formatter outputs both before and after change data for insert, update, and delete operations.

10.2.38.4.5.1 Using the Delimited Text Row Formatter

The Delimited Text Row Formatter is the Delimited Text Formatter that was included a release prior to the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 19.1.0.0.0 release. It writes the after change data for inserts and updates, and before change data for deletes.

- [Message Formatting Details](#)

- [Sample Formatted Messages](#)
- [Output Format Summary Log](#)
- [Configuration](#)
- [Metadata Change Events](#)
- [Additional Considerations](#)

10.2.38.4.5.1.1 Message Formatting Details

The automated output of meta-column fields in generated delimited text messages has been removed as of Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) release 21.1. Meta-column fields can still be output; however, they need to be explicitly configured as the following property:

```
gg.handler.name.format.metaColumnsTemplate
```

To output the metacolumns as in previous versions configure the following:

```
gg.handler.name.format.metaColumnsTemplate=${optype[op_type]},$
{objectname[table]},${timestamp[op_ts]},${currenttimestamp[current_ts]},$
{position[pos]}
```

To also include the primary key columns and the tokens configure as follows:

```
gg.handler.name.format.metaColumnsTemplate=${optype[op_type]},$
{objectname[table]},${timestamp[op_ts]},${currenttimestamp[current_ts]},$
{position[pos]},${primarykeycolumns[primary_keys]},${alltokens[tokens]}
```

For more information, see

see the configuration property `gg.handler.name.format.metaColumnsTemplate` in the [Delimited Text Formatter Configuration Properties](#) table.

Formatting details:

- **Operation Type** : Indicates the type of database operation from the source trail file. Default values are I for insert, U for update, D for delete, T for truncate. Output of this field is suppressible.
- **Fully Qualified Table Name**: The fully qualified table name is the source database table including the catalog name, and the schema name. The format of the fully qualified table name is *catalog_name.schema_name.table_name*. The output of this field is suppressible.
- **Operation Timestamp** : The commit record timestamp from the source system. All operations in a transaction (unbatched transaction) will have the same operation timestamp. This timestamp is fixed, and the operation timestamp is the same if the trail file is replayed. The output of this field is suppressible.
- **Current Timestamp** : The timestamp of the current time when the delimited text formatter processes the current operation record. This timestamp follows the ISO-8601 format and includes microsecond precision. Replaying the trail file does *not* result in the same timestamp for the same operation. The output of this field is suppressible.
- **Trail Position** : The concatenated sequence number and RBA number from the source trail file. The trail position lets you trace the operation back to the source trail file. The sequence number is the source trail file number. The RBA number is the offset in the trail file. The output of this field is suppressible.
- **Tokens** : The token key value pairs from the source trail file. The output of this field in the delimited text output is suppressed unless the `includeTokens` configuration property on the corresponding handler is explicitly set to true.

10.2.38.4.5.1.2 Sample Formatted Messages

The following sections contain sample messages from the Delimited Text Formatter. The default field delimiter has been changed to a pipe character, |, to more clearly display the message.

- [Sample Insert Message](#)
- [Sample Update Message](#)
- [Sample Delete Message](#)
- [Sample Truncate Message](#)

10.2.38.4.5.1.2.1 Sample Insert Message

```
I|GG.TCUSTORD|2013-06-02
22:14:36.000000|2015-09-18T13:23:01.612001|00000000000000001444|R=AADPkVAAEAAEqL2A
AA|WILL|1994-09-30:15:33:00|CAR|144|17520.00|3|100
```

10.2.38.4.5.1.2.2 Sample Update Message

```
U|GG.TCUSTORD|2013-06-02
22:14:41.000000|2015-09-18T13:23:01.987000|00000000000000002891|R=AADPkVAAEAAEqL2A
AA|BILL|1995-12-31:15:00:00|CAR|765|14000.00|3|100
```

10.2.38.4.5.1.2.3 Sample Delete Message

```
D,GG.TCUSTORD,2013-06-02
22:14:41.000000,2015-09-18T13:23:02.000000,00000000000000004338,L=206080450,6=9.0.
80330,R=AADPkVAAEAAEqLZAAC,DAVE,1993-11-03:07:51:35,PLANE,600,, ,
```

10.2.38.4.5.1.2.4 Sample Truncate Message

```
T|GG.TCUSTORD|2013-06-02
22:14:41.000000|2015-09-18T13:23:02.001000|00000000000000004515|R=AADPkVAAEAAEqL2A
AB|||||
```

10.2.38.4.5.1.3 Output Format Summary Log

If INFO level logging is enabled, the Java log4j logging logs a summary of the delimited text output format. A summary of the delimited fields is logged for each source table encountered and occurs when the first operation for that table is received by the Delimited Text formatter. This detailed explanation of the fields of the delimited text output may be useful when you perform an initial setup. When a metadata change event occurs, the summary of the delimited fields is regenerated and logged again at the first subsequent operation for that table.

10.2.38.4.5.1.4 Configuration

- [Review a Sample Configuration](#)

10.2.38.4.5.1.4.1 Review a Sample Configuration

The following is a sample configuration for the Delimited Text formatter in the Java Adapter configuration file:

```
gg.handler.name.format.includeColumnNames=false
gg.handler.name.format.insertOpKey=I
gg.handler.name.format.updateOpKey=U
gg.handler.name.format.deleteOpKey=D
gg.handler.name.format.truncateOpKey=T
gg.handler.name.format.encoding=UTF-8
gg.handler.name.format.fieldDelimiter=CDATA[\u0001]
gg.handler.name.format.lineDelimiter=CDATA[\n]
gg.handler.name.format.keyValueDelimiter=CDATA[=]
gg.handler.name.format.keyValuePairDelimiter=CDATA[,]
gg.handler.name.format.pkUpdateHandling=abend
```

```
gg.handler.name.format.nullValueRepresentation=NULL  
gg.handler.name.format.missingValueRepresentation=CDATA[]  
gg.handler.name.format.includeGroupCols=false  
gg.handler.name.format=delimitedtext
```

10.2.38.4.5.1.5 Metadata Change Events

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) now handles metadata change events at runtime. This assumes that the replicated database and upstream replication processes are propagating metadata change events. The Delimited Text Formatter changes the output format to accommodate the change and the Delimited Text Formatter continue running.

Note

A metadata change may affect downstream applications. Delimited text formats include a fixed number of fields that are positionally relevant. Deleting a column in the source table can be handled seamlessly during Oracle GoldenGate runtime, but results in a change in the total number of fields, and potentially changes the positional relevance of some fields. Adding an additional column or columns is probably the least impactful metadata change event, assuming that the new column is added to the end. Consider the impact of a metadata change event before executing the event. When metadata change events are frequent, Oracle recommends that you consider a more flexible and self-describing format, such as JSON or XML.

10.2.38.4.5.1.6 Additional Considerations

Exercise care when you choose field and line delimiters. It is important to choose delimiter values that will not occur in the content of the data.

The Java Adapter configuration trims leading and trailing characters from configuration values when they are determined to be whitespace. However, you may want to choose field delimiters, line delimiters, null value representations, and missing value representations that include or are fully considered to be whitespace. In these cases, you must employ specialized syntax in the Java Adapter configuration file to preserve the whitespace. To preserve the whitespace, when your configuration values contain leading or trailing characters that are considered whitespace, wrap the configuration value in a CDATA[] wrapper. For example, a configuration value of \n should be configured as CDATA[\n].

You can use regular expressions to search column values then replace matches with a specified value. You can use this search and replace functionality together with the Delimited Text Formatter to ensure that there are no collisions between column value contents and field and line delimiters. For more information, see [Using Regular Expression Search and Replace](#).

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) applications store data differently from RDBMSs. Update and delete operations in an RDBMS result in a change to the existing data. However, in GG for DAA applications, data is appended instead of changed. Therefore, the current state of a given row consolidates all of the existing operations for that row in the HDFS system. This leads to some special scenarios as described in the following sections.

- [Primary Key Updates](#)
- [Data Consolidation](#)

10.2.38.4.5.1.6.1 Primary Key Updates

In Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) integrations, primary key update operations require special consideration and planning. Primary key updates modify one or more of the primary keys for the given row from the source database.

Because data is appended in GG for DAA applications, a primary key update operation looks more like an insert than an update without any special handling. You can configure how the Delimited Text formatter handles primary key updates. These are the configurable behaviors:

Table 10-56 Configurable Behavior

Value	Description
abend	By default the delimited text formatter terminates in the case of a primary key update.
update	The primary key update is treated like any other update operation. Use this configuration alternative only if you can guarantee that the primary key is not used as selection criteria to select row data from a GG for DAA system.
delete-insert	The primary key update is treated as a special case of a delete, using the before-image data and an insert using the after-image data. This configuration may more accurately model the effect of a primary key update in a GG for DAA application. However, if this configuration is selected it is important to have full supplemental logging enabled on replication at the source database. Without full supplemental logging, the delete operation will be correct, but the insert operation will not contain all of the data for all of the columns for a full representation of the row data in the GG for DAA application.

10.2.38.4.5.1.6.2 Data Consolidation

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) applications append data to the underlying storage. Analytic tools generally spawn MapReduce programs that traverse the data files and consolidate all the operations for a given row into a single output. Therefore, it is important to specify the order of operations. The Delimited Text formatter provides a number of metadata fields to do this. The operation timestamp may be sufficient to fulfill this requirement. Alternatively, the current timestamp may be the best indicator of the order of operations. In this situation, the trail position can provide a tie-breaking field on the operation timestamp. Lastly, the current timestamp may provide the best indicator of order of operations in GG for DAA.

10.2.38.4.5.2 Delimited Text Operation Formatter

The Delimited Text Operation Formatter outputs both before and after change data for insert, update, and delete operations.

- [Message Formatting Details](#)
- [Sample Formatted Messages](#)
- [Output Format Summary Log](#)
- [Delimited Text Formatter Configuration Properties](#)
- [Review a Sample Configuration](#)
- [Metadata Change Events](#)

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) now handles metadata change events at runtime. This assumes that the replicated database and upstream replication processes are propagating metadata change events. The Delimited Text Formatter changes the output format to accommodate the change and the Delimited Text Formatter continue running.

- [Additional Considerations](#)

Exercise care when you choose field and line delimiters. It is important to choose delimiter values that do not occur in the content of the data.

10.2.38.4.5.2.1 Message Formatting Details

The automated output of meta-column fields in generated delimited text messages has been removed as of Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) release 21.1. Meta-column fields can still be output; however, they need to be explicitly configured as the following property: `gg.handler.name.format.metaColumnsTemplate`. For more information, see the configuration property `gg.handler.name.format.metaColumnsTemplate` in the [Delimited Text Formatter Configuration Properties](#) table.

To output the metacolumns as in previous versions configure the following:

```
gg.handler.name.format.metaColumnsTemplate=${optype[op_type]},$
{objectname[table]},${timestamp[op_ts]},${currenttimestamp[current_ts]},$
{position[pos]}
```

To also include the primary key columns and the tokens configure as follows:

```
gg.handler.name.format.metaColumnsTemplate=${optype[op_type]},$
{objectname[table]},${timestamp[op_ts]},${currenttimestamp[current_ts]},$
{position[pos]},${primarykeycolumns[primary_keys]},${alltokens[tokens]}
```

Formatting details:

- **Operation Type** : Indicates the type of database operation from the source trail file. Default values are I for insert, U for update, D for delete, T for truncate. Output of this field is suppressible.
- **Fully Qualified Table Name** : The fully qualified table name is the source database table including the catalog name, and the schema name. The format of the fully qualified table name is `catalog_name.schema_name.table_name`. The output of this field is suppressible.
- **Operation Timestamp** : The commit record timestamp from the source system. All operations in a transaction (unbatched transaction) will have the same operation timestamp. This timestamp is fixed, and the operation timestamp is the same if the trail file is replayed. The output of this field is suppressible.
- **Current Timestamp** : The timestamp of the current time when the delimited text formatter processes the current operation record. This timestamp follows the ISO-8601 format and includes microsecond precision. Replaying the trail file does not result in the same timestamp for the same operation. The output of this field is suppressible.
- **Trail Position** : The concatenated sequence number and RBA number from the source trail file. The trail position lets you trace the operation back to the source trail file. The sequence number is the source trail file number. The RBA number is the offset in the trail file. The output of this field is suppressible.
- **Tokens** : The token key value pairs from the source trail file. The output of this field in the delimited text output is suppressed unless the `includeTokens` configuration property on the corresponding handler is explicitly set to true.

10.2.38.4.5.2.2 Sample Formatted Messages

The following sections contain sample messages from the Delimited Text Formatter. The default field delimiter has been changed to a pipe character, `|`, to more clearly display the message.

- [Sample Insert Message](#)
- [Sample Update Message](#)
- [Sample Delete Message](#)
- [Sample Truncate Message](#)

10.2.38.4.5.2.2.1 Sample Insert Message

```
I|GG.TCUSTMER|2015-11-05 18:45:36.000000|2019-04-17T04:49:00.156000|
00000000000000001956|R=AAKifQAAKAAAFDHAAA,t=L=7824137832,6=2.3.228025||WILL||BG
SOFTWARE CO.||SEATTLE||WA
```

10.2.38.4.5.2.2.2 Sample Update Message

```
U|QASOURCE.TCUSTMER|2015-11-05
18:45:39.000000|2019-07-16T11:54:06.008002|00000000000000005100|R=AAKifQAAKAAAFDHAAE|ANN|
ANN|ANN'S
BOATS||SEATTLE|NEW YORK|WA|NY
```

10.2.38.4.5.2.2.3 Sample Delete Message

```
D|QASOURCE.TCUSTORD|2015-11-05 18:45:39.000000|2019-07-16T11:54:06.009000|
00000000000000005272|L=7824137921,R=AAKifSAAKAAAMZHAAE,6=9.9.479055|DAVE||
1993-11-03 07:51:35||PLANE||600||135000.00||2||200|
```

10.2.38.4.5.2.2.4 Sample Truncate Message

```
T|QASOURCE.TCUSTMER|2015-11-05 18:45:39.000000|2019-07-16T11:54:06.004002|
00000000000000003600|R=AAKifQAAKAAAFDHAAE|||||
```

10.2.38.4.5.2.3 Output Format Summary Log

If INFO level logging is enabled, the Java log4j logging logs a summary of the delimited text output format. A summary of the delimited fields is logged for each source table encountered and occurs when the first operation for that table is received by the Delimited Text formatter. This detailed explanation of the fields of the delimited text output may be useful when you perform an initial setup. When a metadata change event occurs, the summary of the delimited fields is regenerated and logged again at the first subsequent operation for that table.

10.2.38.4.5.2.4 Delimited Text Formatter Configuration Properties

Table 10-57 Delimited Text Formatter Configuration Properties

Properties	Option al / Requir ed	Legal Values	Default	Explanation
gg.handler.name.format	Requir ed	delimite dtext_op	None	Selects the Delimited Text Operation Formatter as the formatter.
gg.handler.name.format.includeC olumnNames	Option al	true false	false	Controls the output of writing the column names as a delimited field preceding the column value. When true, the output resembles: COL1_Name COL1_Before_Value COL1_After_Value COL2_Name COL2_Before_Value COL2_After_Value When false, the output resembles: COL1_Before_Value COL1_After_Value COL2_Before_Value COL2_After_Value
gg.handler.name.format.disableE scaping	Option al	true false	false	Set to true to disable the escaping of characters which conflict with the configured delimiters. Ensure that it is set to true if gg.handler.name.format.fieldDelimi ter is set to a value of multiple characters.

Table 10-57 (Cont.) Delimited Text Formatter Configuration Properties

Properties	Option al / Requir ed	Legal Values	Default	Explanation
<code>gg.handler.name.format.insertOpKey</code>	Option al	Any string	I	Indicator to be inserted into the output record to indicate an insert operation.
<code>gg.handler.name.format.updateOpKey</code>	Option al	Any string	U	Indicator to be inserted into the output record to indicate an update operation.
<code>gg.handler.name.format.deleteOpKey</code>	Option al	Any string	D	Indicator to be inserted into the output record to indicate a delete operation.
<code>gg.handler.name.format.truncateOpKey</code>	Option al	Any string	T	Indicator to be inserted into the output record to indicate a truncate operation.
<code>gg.handler.name.format.encoding</code>	Option al	Any encoding name or alias supported by Java.	The native system encoding of the machine hosting the Oracle GoldenGate process.	Determines the encoding of the output delimited text.
<code>gg.handler.name.format.fieldDelimiter</code>	Option al	Any String	ASCII 001 (the default Hive delimiter)	The delimiter used between delimited fields. This value supports CDATA[] wrapping. If a delimiter of more than one character is configured, then escaping is automatically disabled.
<code>gg.handler.name.format.lineDelimiter</code>	Option al	Any String	Newline (the default Hive delimiter)	The delimiter used between delimited fields. This value supports CDATA[] wrapping.
<code>gg.handler.name.format.keyValueDelimiter</code>	Option al	Any string	=	Specifies a delimiter between keys and values in a map. Key1=value1. Tokens are mapped values. Configuration value supports CDATA[] wrapping.
<code>gg.handler.name.format.keyValuePairDelimiter</code>	Option al	Any string	,	Specifies a delimiter between key value pairs in a map. Key1=Value1,Key2=Value2. Tokens are mapped values. Configuration value supports CDATA[] wrapping.
<code>gg.handler.name.format.nullValueRepresentation</code>	Option al	Any string	NULL	Specifies what is included in the delimited output in the case of a NULL value. Configuration value supports CDATA[] wrapping.
<code>gg.handler.name.format.missingValueRepresentation</code>	Option al	Any string	""(no value)	Specifies what is included in the delimited text output in the case of a missing value. Configuration value supports CDATA[] wrapping.
<code>gg.handler.name.format.includeMetadataColumnNames</code>	Option al	true false	false	Set to true, a field is included prior to each metadata column value, which is the column name of the metadata column. You can use it to make delimited messages more self-describing.

Table 10-57 (Cont.) Delimited Text Formatter Configuration Properties

Properties	Option al / Requir ed	Legal Values	Default	Explanation
<code>gg.handler.name.format.wrapStringsInQuotes</code>	Option al	true false	false	Set to true to wrap string value output in the delimited text format in double quotes (").
<code>gg.handler.name.format.includeGroupCols</code>	Option al	true false	false	If set to true, the columns are grouped into sets of all names, all before values, and all after values U, QASOURCE.TCUSTMER, 2015-11-05 18:45:39.000000, 2019-04-17T05:19:30.556000, 000000000000000005100, R=AAKifQAAKAAAFDHAAE, CUST_CODE, NAME, CITY, STATE, ANN, ANN'S BOATS, SEATTLE, WA, ANN, , NEW YORK, NY
<code>gg.handler.name.format.enableFieldDescriptorHeaders</code>	Option al	true false	false	Set to true to add a descriptive header to each data file for delimited text output. The header will be the individual field names separated by the field delimiter.
<code>gg.handler.name.format.metaColumnTemplate</code>	Option al	See Metacolumn Keywords	None	The current meta column information can be configured in a simple manner and removes the explicit need to use: <pre>insertOpKey updateOpKey deleteOpKey truncateOpKey includeTableName includeOpTimestamp includeOpType includePosition includeCurrentTimestamp, useIso8601Format</pre> It is a comma-delimited string consisting of one or more templated values that represent the template. For more information about the Metacolumn keywords, see Metacolumn Keywords. This is an example that would produce a list of metacolumns: \$ {optype}, \${token.ROWID}, \$ {sys.username}, \$ {currenttimestamp}

10.2.38.4.5.2.5 Review a Sample Configuration

The following is a sample configuration for the Delimited Text formatter in the Java Adapter configuration file:

```
gg.handler.name.format.includeColumnNames=false
gg.handler.name.format.insertOpKey=I
gg.handler.name.format.updateOpKey=U
gg.handler.name.format.deleteOpKey=D
```



```
gg.handler.name.format.truncateOpKey=T
gg.handler.name.format.encoding=UTF-8
gg.handler.name.format.fieldDelimiter=CDATA[\u0001]
gg.handler.name.format.lineDelimiter=CDATA[\n]
gg.handler.name.format.keyValueDelimiter=CDATA[=]
gg.handler.name.format.keyValuePairDelimiter=CDATA[,]
gg.handler.name.format.nullValueRepresentation=NULL
gg.handler.name.format.missingValueRepresentation=CDATA[]
gg.handler.name.format.includeGroupCols=false
gg.handler.name.format=delimitedtext_op
```

10.2.38.4.5.2.6 Metadata Change Events

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) now handles metadata change events at runtime. This assumes that the replicated database and upstream replication processes are propagating metadata change events. The Delimited Text Formatter changes the output format to accommodate the change and the Delimited Text Formatter continue running.

Note

A metadata change may affect downstream applications. Delimited text formats include a fixed number of fields that are positionally relevant. Deleting a column in the source table can be handled seamlessly during Oracle GoldenGate runtime, but results in a change in the total number of fields, and potentially changes the positional relevance of some fields. Adding an additional column or columns is probably the least impactful metadata change event, assuming that the new column is added to the end. Consider the impact of a metadata change event before executing the event. When metadata change events are frequent, Oracle recommends that you consider a more flexible and self-describing format, such as JSON or XML.

10.2.38.4.5.2.7 Additional Considerations

Exercise care when you choose field and line delimiters. It is important to choose delimiter values that do not occur in the content of the data.

The Java Adapter configuration trims leading and trailing characters from configuration values when they are determined to be whitespace. However, you may want to choose field delimiters, line delimiters, null value representations, and missing value representations that include or are fully considered to be whitespace. In these cases, you must employ specialized syntax in the Java Adapter configuration file to preserve the whitespace. To preserve the whitespace, when your configuration values contain leading or trailing characters that are considered whitespace, wrap the configuration value in a CDATA[] wrapper. For example, a configuration value of \n should be configured as CDATA[\n].

You can use regular expressions to search column values then replace matches with a specified value. You can use this search and replace functionality together with the Delimited Text Formatter to ensure that there are no collisions between column value contents and field and line delimiters. For more information, see [Using Regular Expression Search and Replace](#).

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) applications store data differently from RDBMSs. Update and delete operations in an RDBMS result in a change to the existing data. However, in GG for DAA, data is appended instead of changed. Therefore, the current state of a given row consolidates all of the existing operations for that row in the HDFS system. This leads to some special scenarios as described in the following sections.

10.2.38.4.6 Using the JSON Formatter

The JavaScript Object Notation (JSON) formatter can output operations from the source trail file in either row-based format or operation-based format. It formats operation data from the source trail file into a JSON objects. Each insert, update, delete, and truncate operation is formatted into an individual JSON message.

- [Operation Metadata Formatting Details](#)
- [Operation Data Formatting Details](#)
- [Row Data Formatting Details](#)
- [Sample JSON Messages](#)
- [JSON Schemas](#)
- [JSON Formatter Configuration Properties](#)
- [Review a Sample Configuration](#)
- [Metadata Change Events](#)
- [JSON Primary Key Updates](#)
- [Integrating Oracle Stream Analytics](#)
- [Mongo Document Formatting Details](#)

10.2.38.4.6.1 Operation Metadata Formatting Details

To output the metacolumns configure the following:

```
gg.handler.name.format.metaColumnsTemplate=${objectname[table]},${
  optype[op_type]},${timestamp[op_ts]},${currenttimestamp[current_ts]},$
  position[pos]}
```

To also include the primary key columns and the tokens configure as follows:

```
gg.handler.name.format.metaColumnsTemplate=${objectname[table]},$
  optype[op_type]},${timestamp[op_ts]},${currenttimestamp[current_ts]},$
  position[pos]},${primarykeycolumns[primary_keys]},${alltokens[tokens]}
```

For more information see the configuration property:

```
gg.handler.name.format.metaColumnsTemplate.
```

10.2.38.4.6.2 Operation Data Formatting Details

JSON messages begin with the operation metadata fields, which are followed by the operation data fields. This data is represented by before and after members that are objects. These objects contain members whose keys are the column names and whose values are the column values.

Operation data is modeled as follows:

- Inserts: Includes the after-image data.
- Updates: Includes both the before-image and the after-image data.
- Deletes: Includes the before-image data.

Column values for an operation from the source trail file can have one of three states: the column has a value, the column value is null, or the column value is missing. The JSON Formatter maps these column value states into the created JSON objects as follows:

- The column has a value: The column value is output. In the following example, the member STATE has a value.

```
"after":{      "CUST_CODE":"BILL",      "NAME":"BILL'S USED CARS",
"CITY":"DENVER",      "STATE":"CO"    }
```

- The column value is null: The default output value is a JSON NULL. In the following example, the member STATE is null.

```
"after":{      "CUST_CODE":"BILL",      "NAME":"BILL'S USED CARS",
"CITY":"DENVER",      "STATE":null    }
```

- The column value is missing: The JSON contains no element for a missing column value. In the following example, the member STATE is missing.

```
"after":{      "CUST_CODE":"BILL",      "NAME":"BILL'S USED CARS",
"CITY":"DENVER",    }
```

The default setting of the JSON Formatter is to map the data types from the source trail file to the associated JSON data type. JSON supports few data types, so this functionality usually results in the mapping of numeric fields from the source trail file to members typed as numbers. This data type mapping can be configured treat all data as strings.

10.2.38.4.6.3 Row Data Formatting Details

JSON messages begin with the operation metadata fields, which are followed by the operation data fields. For row data formatting, this are the source column names and source column values as JSON key value pairs. This data is represented by before and after members that are objects. These objects contain members whose keys are the column names and whose values are the column values.

Row data is modeled as follows:

- Inserts: Includes the after-image data.
- Updates: Includes the after-image data.
- Deletes: Includes the before-image data.

Column values for an operation from the source trail file can have one of three states: the column has a value, the column value is null, or the column value is missing. The JSON Formatter maps these column value states into the created JSON objects as follows:

- The column has a value: The column value is output. In the following example, the member STATE has a value.

```
"CUST_CODE":"BILL",      "NAME":"BILL'S USED CARS",
"CITY":"DENVER",      "STATE":"CO"    }
```

- The column value is null :The default output value is a JSON NULL. In the following example, the member STATE is null.

```
"CUST_CODE":"BILL",      "NAME":"BILL'S USED CARS",
"CITY":"DENVER",      "STATE":null    }
```

- The column value is missing: The JSON contains no element for a missing column value. In the following example, the member STATE is missing.

```
"CUST_CODE":"BILL",      "NAME":"BILL'S USED CARS",
"CITY":"DENVER",    }
```

The default setting of the JSON Formatter is to map the data types from the source trail file to the associated JSON data type. JSON supports few data types, so this functionality usually results in the mapping of numeric fields from the source trail file to members typed as numbers. This data type mapping can be configured to treat all data as strings.

10.2.38.4.6.4 Sample JSON Messages

The following topics are sample JSON messages created by the JSON Formatter for insert, update, delete, and truncate operations.

- [Sample Operation Modeled JSON Messages](#)
- [Sample Flattened Operation Modeled JSON Messages](#)
- [Sample Row Modeled JSON Messages](#)
- [Sample Primary Key Output JSON Message](#)

10.2.38.4.6.4.1 Sample Operation Modeled JSON Messages

Insert

```
{
  "table": "QASOURCE.TCUSTORD",
  "op_type": "I",
  "op_ts": "2015-11-05 18:45:36.000000",
  "current_ts": "2016-10-05T10:15:51.267000",
  "pos": "00000000000000002928",
  "after": {
    "CUST_CODE": "WILL",
    "ORDER_DATE": "1994-09-30:15:33:00",
    "PRODUCT_CODE": "CAR",
    "ORDER_ID": 144,
    "PRODUCT_PRICE": 17520.00,
    "PRODUCT_AMOUNT": 3,
    "TRANSACTION_ID": 100
  }
}
```

Update

```
{
  "table": "QASOURCE.TCUSTORD",
  "op_type": "U",
  "op_ts": "2015-11-05 18:45:39.000000",
  "current_ts": "2016-10-05T10:15:51.310002",
  "pos": "00000000000000004300",
  "before": {
    "CUST_CODE": "BILL",
    "ORDER_DATE": "1995-12-31:15:00:00",
    "PRODUCT_CODE": "CAR",
    "ORDER_ID": 765,
    "PRODUCT_PRICE": 15000.00,
    "PRODUCT_AMOUNT": 3,
    "TRANSACTION_ID": 100
  },
  "after": {
    "CUST_CODE": "BILL",
    "ORDER_DATE": "1995-12-31:15:00:00",
    "PRODUCT_CODE": "CAR",
    "ORDER_ID": 765,
    "PRODUCT_PRICE": 14000.00
  }
}
```

```
    }
  }
```

Delete

```
{
  "table": "QASOURCE.TCUSTORD",
  "op_type": "D",
  "op_ts": "2015-11-05 18:45:39.000000",
  "current_ts": "2016-10-05T10:15:51.312000",
  "pos": "000000000000000005272",
  "before": {
    "CUST_CODE": "DAVE",
    "ORDER_DATE": "1993-11-03:07:51:35",
    "PRODUCT_CODE": "PLANE",
    "ORDER_ID": 600,
    "PRODUCT_PRICE": 135000.00,
    "PRODUCT_AMOUNT": 2,
    "TRANSACTION_ID": 200
  }
}
```

Truncate

```
{
  "table": "QASOURCE.TCUSTORD",
  "op_type": "T",
  "op_ts": "2015-11-05 18:45:39.000000",
  "current_ts": "2016-10-05T10:15:51.312001",
  "pos": "000000000000000005480",
}
```

10.2.38.4.6.4.2 Sample Flattened Operation Modeled JSON Messages**Insert**

```
{
  "table": "QASOURCE.TCUSTORD",
  "op_type": "I",
  "op_ts": "2015-11-05 18:45:36.000000",
  "current_ts": "2016-10-05T10:34:47.956000",
  "pos": "000000000000000002928",
  "after.CUST_CODE": "WILL",
  "after.ORDER_DATE": "1994-09-30:15:33:00",
  "after.PRODUCT_CODE": "CAR",
  "after.ORDER_ID": 144,
  "after.PRODUCT_PRICE": 17520.00,
  "after.PRODUCT_AMOUNT": 3,
  "after.TRANSACTION_ID": 100
}
```

Update

```
{
  "table": "QASOURCE.TCUSTORD",
  "op_type": "U",
  "op_ts": "2015-11-05 18:45:39.000000",
  "current_ts": "2016-10-05T10:34:48.192000",
  "pos": "00000000000000004300",
  "before.CUST_CODE": "BILL",
  "before.ORDER_DATE": "1995-12-31:15:00:00",
  "before.PRODUCT_CODE": "CAR",
  "before.ORDER_ID": 765,
  "before.PRODUCT_PRICE": 15000.00,
  "before.PRODUCT_AMOUNT": 3,
  "before.TRANSACTION_ID": 100,
  "after.CUST_CODE": "BILL",
  "after.ORDER_DATE": "1995-12-31:15:00:00",
  "after.PRODUCT_CODE": "CAR",
  "after.ORDER_ID": 765,
  "after.PRODUCT_PRICE": 14000.00
}
```

Delete

```
{
  "table": "QASOURCE.TCUSTORD",
  "op_type": "D",
  "op_ts": "2015-11-05 18:45:39.000000",
  "current_ts": "2016-10-05T10:34:48.193000",
  "pos": "00000000000000005272",
  "before.CUST_CODE": "DAVE",
  "before.ORDER_DATE": "1993-11-03:07:51:35",
  "before.PRODUCT_CODE": "PLANE",
  "before.ORDER_ID": 600,
  "before.PRODUCT_PRICE": 135000.00,
  "before.PRODUCT_AMOUNT": 2,
  "before.TRANSACTION_ID": 200
}
```

Truncate

```
{
  "table": "QASOURCE.TCUSTORD",
  "op_type": "D",
  "op_ts": "2015-11-05 18:45:39.000000",
  "current_ts": "2016-10-05T10:34:48.193001",
  "pos": "00000000000000005480",
  "before.CUST_CODE": "JANE",
  "before.ORDER_DATE": "1995-11-11:13:52:00",
  "before.PRODUCT_CODE": "PLANE",
  "before.ORDER_ID": 256,
  "before.PRODUCT_PRICE": 133300.00,
  "before.PRODUCT_AMOUNT": 1,
}
```

```
    "before.TRANSACTION_ID":100
  }
```

10.2.38.4.6.4.3 Sample Row Modeled JSON Messages

Insert

```
{
  "table":"QASOURCE.TCUSTORD",
  "op_type":"I",
  "op_ts":"2015-11-05 18:45:36.000000",
  "current_ts":"2016-10-05T11:10:42.294000",
  "pos":"00000000000000002928",
  "CUST_CODE":"WILL",
  "ORDER_DATE":"1994-09-30:15:33:00",
  "PRODUCT_CODE":"CAR",
  "ORDER_ID":144,
  "PRODUCT_PRICE":17520.00,
  "PRODUCT_AMOUNT":3,
  "TRANSACTION_ID":100
}
```

Update

```
{
  "table":"QASOURCE.TCUSTORD",
  "op_type":"U",
  "op_ts":"2015-11-05 18:45:39.000000",
  "current_ts":"2016-10-05T11:10:42.350005",
  "pos":"00000000000000004300",
  "CUST_CODE":"BILL",
  "ORDER_DATE":"1995-12-31:15:00:00",
  "PRODUCT_CODE":"CAR",
  "ORDER_ID":765,
  "PRODUCT_PRICE":14000.00
}
```

Delete

```
{
  "table":"QASOURCE.TCUSTORD",
  "op_type":"D",
  "op_ts":"2015-11-05 18:45:39.000000",
  "current_ts":"2016-10-05T11:10:42.351002",
  "pos":"00000000000000005272",
  "CUST_CODE":"DAVE",
  "ORDER_DATE":"1993-11-03:07:51:35",
  "PRODUCT_CODE":"PLANE",
  "ORDER_ID":600,
  "PRODUCT_PRICE":135000.00,
  "PRODUCT_AMOUNT":2,
  "TRANSACTION_ID":200
}
```

Truncate

```
{
  "table": "QASOURCE.TCUSTORD",
  "op_type": "T",
  "op_ts": "2015-11-05 18:45:39.000000",
  "current_ts": "2016-10-05T11:10:42.351003",
  "pos": "00000000000000005480",
}
```

10.2.38.4.6.4.4 Sample Primary Key Output JSON Message

```
{
  "table": "DDL_OGGSRC.TCUSTMER",
  "op_type": "I",
  "op_ts": "2015-10-26 03:00:06.000000",
  "current_ts": "2016-04-05T08:59:23.001000",
  "pos": "00000000000000006605",
  "primary_keys": [
    "CUST_CODE"
  ],
  "after": {
    "CUST_CODE": "WILL",
    "NAME": "BG SOFTWARE CO.",
    "CITY": "SEATTLE",
    "STATE": "WA"
  }
}
```

10.2.38.4.6.5 JSON Schemas

By default, JSON schemas are generated for each source table encountered. JSON schemas are generated on a just in time basis when an operation for that table is first encountered. Newer schemas are generated when there is a change in the metadata. A JSON schema is not required to parse a JSON object. However, many JSON parsers can use a JSON schema to perform a validating parse of a JSON object. Alternatively, you can review the JSON schemas to understand the layout of output JSON objects. By default, the JSON schemas are created in the *GoldenGate_Home/dirdef* directory and are named by the following convention:

FULLY_QUALIFIED_TABLE_NAME.schema.json

The generation of the JSON schemas is suppressible.

The following JSON schema example is for the JSON object listed in [Sample Operation Modeled JSON Messages](#).

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "QASOURCE.TCUSTORD",
  "description": "JSON schema for table QASOURCE.TCUSTORD",
  "definitions": {
    "row": {
      "type": "object",
      "properties": {
        "CUST_CODE": {
```



```

        "type":[
            "string",
            "null"
        ]
    },
    "ORDER_DATE":{
        "type":[
            "string",
            "null"
        ]
    },
    "PRODUCT_CODE":{
        "type":[
            "string",
            "null"
        ]
    },
    "ORDER_ID":{
        "type":[
            "number",
            "null"
        ]
    },
    "PRODUCT_PRICE":{
        "type":[
            "number",
            "null"
        ]
    },
    "PRODUCT_AMOUNT":{
        "type":[
            "integer",
            "null"
        ]
    },
    "TRANSACTION_ID":{
        "type":[
            "number",
            "null"
        ]
    }
},
"additionalProperties":false
},
"tokens":{
    "type":"object",
    "description":"Token keys and values are free form key value
pairs.",
    "properties":{
    },
    "additionalProperties":true
}
},
"type":"object",

```

```

    "properties":{
      "table":{
        "description":"The fully qualified table name",
        "type":"string"
      },
      "op_type":{
        "description":"The operation type",
        "type":"string"
      },
      "op_ts":{
        "description":"The operation timestamp",
        "type":"string"
      },
      "current_ts":{
        "description":"The current processing timestamp",
        "type":"string"
      },
      "pos":{
        "description":"The position of the operation in the data source",
        "type":"string"
      },
      "primary_keys":{
        "description":"Array of the primary key column names.",
        "type":"array",
        "items":{
          "type":"string"
        },
        "minItems":0,
        "uniqueItems":true
      },
      "tokens":{
        "$ref":"#/definitions/tokens"
      },
      "before":{
        "$ref":"#/definitions/row"
      },
      "after":{
        "$ref":"#/definitions/row"
      }
    },
    "required":[
      "table",
      "op_type",
      "op_ts",
      "current_ts",
      "pos"
    ],
    "additionalProperties":false
  }

```

The following JSON schema example is for the JSON object listed in [Sample Flattened Operation Modeled JSON Messages](#).

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "QASOURCE.TCUSTORD",
  "description": "JSON schema for table QASOURCE.TCUSTORD",
  "definitions": {
    "tokens": {
      "type": "object",
      "description": "Token keys and values are free form key value
pairs.",
      "properties": {
      },
      "additionalProperties": true
    }
  },
  "type": "object",
  "properties": {
    "table": {
      "description": "The fully qualified table name",
      "type": "string"
    },
    "op_type": {
      "description": "The operation type",
      "type": "string"
    },
    "op_ts": {
      "description": "The operation timestamp",
      "type": "string"
    },
    "current_ts": {
      "description": "The current processing timestamp",
      "type": "string"
    },
    "pos": {
      "description": "The position of the operation in the data source",
      "type": "string"
    },
    "primary_keys": {
      "description": "Array of the primary key column names.",
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 0,
      "uniqueItems": true
    },
    "tokens": {
      "$ref": "#/definitions/tokens"
    },
    "before.CUST_CODE": {
      "type": [
        "string",
```

```
        "null"
      ]
    },
    "before.ORDER_DATE":{
      "type":[
        "string",
        "null"
      ]
    },
    "before.PRODUCT_CODE":{
      "type":[
        "string",
        "null"
      ]
    },
    "before.ORDER_ID":{
      "type":[
        "number",
        "null"
      ]
    },
    "before.PRODUCT_PRICE":{
      "type":[
        "number",
        "null"
      ]
    },
    "before.PRODUCT_AMOUNT":{
      "type":[
        "integer",
        "null"
      ]
    },
    "before.TRANSACTION_ID":{
      "type":[
        "number",
        "null"
      ]
    },
    "after.CUST_CODE":{
      "type":[
        "string",
        "null"
      ]
    },
    "after.ORDER_DATE":{
      "type":[
        "string",
        "null"
      ]
    },
    "after.PRODUCT_CODE":{
      "type":[
        "string",
```

```

        "null"
      ]
    },
    "after.ORDER_ID":{
      "type":[
        "number",
        "null"
      ]
    },
    "after.PRODUCT_PRICE":{
      "type":[
        "number",
        "null"
      ]
    },
    "after.PRODUCT_AMOUNT":{
      "type":[
        "integer",
        "null"
      ]
    },
    "after.TRANSACTION_ID":{
      "type":[
        "number",
        "null"
      ]
    }
  },
  "required":[
    "table",
    "op_type",
    "op_ts",
    "current_ts",
    "pos"
  ],
  "additionalProperties":false
}

```

The following JSON schema example is for the JSON object listed in [Sample Row Modeled JSON Messages](#).

```

{
  "$schema":"http://json-schema.org/draft-04/schema#",
  "title":"QASOURCE.TCUSTORD",
  "description":"JSON schema for table QASOURCE.TCUSTORD",
  "definitions":{
    "tokens":{
      "type":"object",
      "description":"Token keys and values are free form key value
pairs.",
      "properties":{
      },
      "additionalProperties":true
    }
  }
}

```

```

    },
    "type": "object",
    "properties": {
      "table": {
        "description": "The fully qualified table name",
        "type": "string"
      },
      "op_type": {
        "description": "The operation type",
        "type": "string"
      },
      "op_ts": {
        "description": "The operation timestamp",
        "type": "string"
      },
      "current_ts": {
        "description": "The current processing timestamp",
        "type": "string"
      },
      "pos": {
        "description": "The position of the operation in the data source",
        "type": "string"
      },
      "primary_keys": {
        "description": "Array of the primary key column names.",
        "type": "array",
        "items": {
          "type": "string"
        },
        "minItems": 0,
        "uniqueItems": true
      },
      "tokens": {
        "$ref": "#/definitions/tokens"
      },
      "CUST_CODE": {
        "type": [
          "string",
          "null"
        ]
      },
      "ORDER_DATE": {
        "type": [
          "string",
          "null"
        ]
      },
      "PRODUCT_CODE": {
        "type": [
          "string",
          "null"
        ]
      },
      "ORDER_ID": {

```

```

        "type": [
            "number",
            "null"
        ]
    },
    "PRODUCT_PRICE": {
        "type": [
            "number",
            "null"
        ]
    },
    "PRODUCT_AMOUNT": {
        "type": [
            "integer",
            "null"
        ]
    },
    "TRANSACTION_ID": {
        "type": [
            "number",
            "null"
        ]
    }
},
"required": [
    "table",
    "op_type",
    "op_ts",
    "current_ts",
    "pos"
],
"additionalProperties": false
}

```

10.2.38.4.6.6 JSON Formatter Configuration Properties

Table 10-58 JSON Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.format</code>	Optional	<code>json</code> <code>json_row</code>	None	Controls whether the generated JSON output messages are operation modeled or row modeled. Set to <code>json</code> for operation modeled or <code>json_row</code> for row modeled.
<code>gg.handler.name.format.insertOpKey</code>	Optional	Any string	I	Indicator to be inserted into the output record to indicate an insert operation.
<code>gg.handler.name.format.updateOpKey</code>	Optional	Any string	U	Indicator to be inserted into the output record to indicate an update operation.
<code>gg.handler.name.format.deleteOpKey</code>	Optional	Any string	D	Indicator to be inserted into the output record to indicate a delete operation.
<code>gg.handler.name.format.truncateOpKey</code>	Optional	Any string	T	Indicator to be inserted into the output record to indicate a truncate operation.

Table 10-58 (Cont.) JSON Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.format.prettyPrint</code>	Optional	true false	false	Controls the output format of the JSON data. True formats the data with white space for easy reading. False generates more compact output that is difficult to read.
 Note If the value of the property <code>prettyPrint</code> is true, a new-line character is added automatically after the generated document.				
<code>gg.handler.name.format.jsonDelimiter</code>	Optional	Any string	"" (no value)	Inserts a delimiter between generated JSONs so that they can be more easily parsed in a continuous stream of data. Configuration value supports CDATA[] wrapping.
<code>gg.handler.name.format.generateSchema</code>	Optional	true false	true	Controls the generation of JSON schemas for the generated JSON documents. JSON schemas are generated on a table-by-table basis. A JSON schema is not required to parse a JSON document. However, a JSON schema help indicate what the JSON documents look like and can be used for a validating JSON parse.
<code>gg.handler.name.format.schemaDirectory</code>	Optional	Any legal, existing file system path	./dirdef	Controls the output location of generated JSON schemas.
<code>gg.handler.name.format.treatAllColumnsAsStrings</code>	Optional	true false	false	Controls the output typing of generated JSON documents. When false, the formatter attempts to map Oracle GoldenGate types to the corresponding JSON type. When true, all data is treated as strings in the generated JSONs and JSON schemas.
<code>gg.handler.name.format.encoding</code>	Optional	Any legal encoding name or alias supported by Java.	UTF-8 (the JSON default)	Controls the output encoding of generated JSON schemas and documents.

Table 10-58 (Cont.) JSON Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.format.versionSchemas</code>	Optional	true false	false	Controls the version of created schemas. Schema versioning creates a schema with a timestamp in the schema directory on the local file system every time a new schema is created. True enables schema versioning. False disables schema versioning.
<code>gg.handler.name.format.iso8601Format</code>	Optional	true false	true	Controls the format of the current timestamp. The default is the ISO 8601 format. A setting of false removes the "T" between the date and time in the current timestamp, which outputs a single space (" ") instead.
<code>gg.handler.name.format.flatten</code>	Optional	true false	false	Controls sending flattened JSON formatted data to the target entity. Must be set to true for the flatten Delimiter property to work. This property is applicable only to Operation Formatted JSON (<code>gg.handler.name.format=json</code>).
<code>gg.handler.name.format.flattenDelimiter</code>	Optional	Any legal character or character string for a JSON field name.	.	Controls the delimiter for concatenated JSON element names. This property supports CDATA[] wrapping to preserve whitespace. It is only relevant when <code>gg.handler.name.format.flatten</code> is set to true.
<code>gg.handler.name.format.beforeObjectName</code>	Optional	Any legal character or character string for a JSON field name.	Any legal JSON attribute name.	Allows you to set whether the JSON element-before, that contains the change column values, can be renamed. This property is only applicable to Operation Formatted JSON (<code>gg.handler.name.format=json</code>).
<code>gg.handler.name.format.afterObjectName</code>	Optional	Any legal character or character string for a JSON field name.	Any legal JSON attribute name.	Allows you to set whether the JSON element, that contains the after-change column values, can be renamed. This property is only applicable to Operation Formatted JSON (<code>gg.handler.name.format=json</code>).

Table 10-58 (Cont.) JSON Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.format.pkUpdateHandling</code>	Optional	<code>abend</code> <code>update</code> <code>delete-</code> <code>insert</code>	<code>abend</code>	Specifies how the formatter handles update operations that change a primary key. Primary key operations can be problematic for the JSON formatter and you need to specially consider it. You can only use this property in conjunction with the row modeled JSON output messages. This property is only applicable to Row Formatted JSON (<code>gg.handler.name.format=json_row</code>). <code>abend</code> : indicates that the process terminates. <code>update</code>: the process handles the operation as a normal update. <code>delete - insert</code>: the process handles the operation as a delete and an insert. Full supplemental logging must be enabled. Without full before and after row images, the insert data will be incomplete.
<code>gg.handler.name.format.omitNullValues</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	Set to <code>true</code> to omit fields that have null values from being included in the generated JSON output.
<code>gg.handler.name.format.omitNullValuesSpecialUpdateHandling</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	Only applicable if <code>gg.handler.name.format.omitNullValues=true</code> . When set to <code>true</code> , it provides special handling to propagate the null value on the update after image if the before image data is missing or has a value.
<code>gg.handler.name.format.enableJsonArrayOutput</code>	Optional	<code>true</code> <code>false</code>	<code>false</code>	Set to <code>true</code> to nest JSON documents representing the operation data into a JSON array. This works for file output and Kafka messages in transaction mode.

Table 10-58 (Cont.) JSON Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.format. metaColumnsTemplate</code>	Optional	See Metacolumn Keywords	None	The current meta column information can be configured in a simple manner and removes the explicit need to use: <pre>insertOpKey updateOpKey deleteOpKey truncateOpKey includeTableName includeOpTimestamp includeOpType includePosition includeCurrentTimestamp, useIso8601Format</pre> It is a comma-delimited string consisting of one or more templated values that represent the template. For more information about the Metacolumn keywords, see Metacolumn Keywords. This is an example that would produce a list of metacolumns: <code>\${optype}, \$ {token.ROWID}, \${sys.username}, \$ {currenttimestamp}</code>

10.2.38.4.6.7 Review a Sample Configuration

The following is a sample configuration for the JSON Formatter in the Java Adapter configuration file:

```
gg.handler.hdfs.format=json
gg.handler.hdfs.format.insertOpKey=I
gg.handler.hdfs.format.updateOpKey=U
gg.handler.hdfs.format.deleteOpKey=D
gg.handler.hdfs.format.truncateOpKey=T
gg.handler.hdfs.format.prettyPrint=false
gg.handler.hdfs.format.jsonDelimiter=CDATA[]
gg.handler.hdfs.format.generateSchema=true
gg.handler.hdfs.format.schemaDirectory=dirdef
gg.handler.hdfs.format.treatAllColumnsAsStrings=false
```

10.2.38.4.6.8 Metadata Change Events

Metadata change events are handled at runtime. When metadata is changed in a table, the JSON schema is regenerated the next time an operation for the table is encountered. The content of created JSON messages changes to reflect the metadata change. For example, if an additional column is added, the new column is included in created JSON messages after the metadata change event.

10.2.38.4.6.9 JSON Primary Key Updates

When the JSON formatter is configured to model operation data, primary key updates require no special treatment and are treated like any other update. The before and after values reflect the change in the primary key.

When the JSON formatter is configured to model row data, primary key updates must be specially handled. The default behavior is to abend. However, by using `thegg.handler.name.format.pkUpdateHandling` configuration property, you can configure the

JSON formatter to model row data to treat primary key updates as either a regular update or as delete and then insert operations. When you configure the formatter to handle primary key updates as delete and insert operations, Oracle recommends that you configure your replication stream to contain the complete before-image and after-image data for updates. Otherwise, the generated insert operation for a primary key update will be missing data for fields that did not change.

10.2.38.4.6.10 Integrating Oracle Stream Analytics

You can integrate Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) with Oracle Stream Analytics (OSA) by sending operation-modeled JSON messages to the Kafka Handler. This works only when the JSON formatter is configured to output operation-modeled JSON messages.

Because OSA requires flattened JSON objects, a new feature in the JSON formatter generates flattened JSONs. To use this feature, set the `gg.handler.name.format.flatten=false` to `true`. (The default setting is `false`). The following is an example of a flattened JSON file:

```
{
  "table": "QASOURCE.TCUSTMER",
  "op_type": "U",
  "op_ts": "2015-11-05 18:45:39.000000",
  "current_ts": "2016-06-22T13:38:45.335001",
  "pos": "000000000000000005100",
  "before.CUST_CODE": "ANN",
  "before.NAME": "ANN'S BOATS",
  "before.CITY": "SEATTLE",
  "before.STATE": "WA",
  "after.CUST_CODE": "ANN",
  "after.CITY": "NEW YORK",
  "after.STATE": "NY"
}
```

10.2.38.4.6.11 Mongo Document Formatting Details

MongoDB Capture processed documents in trail will have two columns:

- Column 0 as `"_id"`, which identifies a document in a collection.
- Column 1 as `"payload"`, which holds all the columns (fields of a collection).

JSON Mongo Document Formatter formats the MongoDB Capture processed documents into a JSON format with only payload information.

Example

The document from trail received is

```
{"after":{"id":{" \"_id\" :
{ \"$_oid\" : \"65b9f02b80f1c27eb4b498e1\" }
} ", "payload":{" \"_id\" :
{ \"$_oid\" : \"65b9f02b80f1c27eb4b498e1\" }
, \"CUST_CODE\" :
```

```
\ "test2\ ", \ "name\ ": \ "hello world\ ", \ "cost\ ": { \ "$numberDouble\ ":
\ "3000.0\ " } } }
```

Will be written as:

```
{ "data": { \ "_id\ ":
{ \ "$oid\ ": \ "65b9f02b80f1c27eb4b498e1\ " }
, \ "CUST_CODE\ ": \ "test2\ ", \ "name\ ": \ "hello world\ ", \ "cost\ ":
{ \ "$numberDouble\ ": \ "3000.0\ " } } }
```

where id field is removed and column name payload is removed.

JSON MongoDB Document Formatter can be configured to write the data either in JSON EXTENDED format or JSON RELAXED format with payload value.

Required Dependencies

Oracle GoldenGate requires that you use the 4.11.1 bson library with JSON Mongo Document Formatter. You can download this driver from: <https://mvnrepository.com/artifact/org.mongodb/bson/4.11.1>

Maven artifacts for bson-4.11.1 as follows:

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>bson</artifactId>
    <version>4.11.1</version>
</dependency>
```

You must include the path of the bson library in the gg.classpath property.

Example:

```
gg.classpath=./bson-4.11.1.jar
```

JSON MongoDB Document Formatter Configuration Properties

Properties	Required/ Optional	Legal Values	Default	Explanation
gg.handler.name.format	Optional	mongodocument	None	Formats the MongoDB Capture processed documents into a JSON format with only payload information
gg.handler.name.format.jsonMode	Optional	RELAXED/ EXTENDED	RELAXED	MongoDB Document will be represented either in Extended or Relaxed format.

gg.handler.name.format.insertOpKey	Optional	Any string	I	Indicator to be inserted into the output record to indicate an insert operation.
gg.handler.name.format.updateOpKey	Optional	Any string	U	Indicator to be inserted into the output record to indicate an update operation.
gg.handler.name.format.deleteOpKey	Optional	Any string	D	Indicator to be inserted into the output record to indicate a delete operation.
gg.handler.name.format.truncateOpKey	Optional	Any string	T	Indicator to be inserted into the output record to indicate a truncate operation.
gg.handler.name.format.metaColumnsTemplate	Optional	See Metacolumn Keywords	None	The current meta column information can be configured in a simple manner and removes the explicit need to use: insertOpKey updateOpKey deleteOpKey truncateOpKey includeTableName includeOpTimestamp includeOpType includePosition includeCurrentTimestamp, useIso8601Format! It is a comma-delimited string consisting of one or more templated values that represent the template. For more information about the Metacolumn keywords, see Metacolumn Keywords. This is an example that would produce a list of metacolumns: \$ {optype}, \$ {token.ROWID}, \$ {sys.username}, \$ {currenttimestamp}

gg.handler.name.format.encoding	Optional	Any legal encoding name or alias supported by Java.	UTF-8 (the JSON default)	Controls the output encoding of generated JSON schemas and documents.

Review a Sample Configuration

The following is a sample configuration for the JSON Mongo Document Formatter in the Java Adapter configuration file:

```
gg.handler.kafka.format=mongodocument

gg.handler.kafka.format.insertOpKey=I

gg.handler.kafka.format.updateOpKey=U

gg.handler.kafka.format.deleteOpKey=D

gg.handler.kafka.format.truncateOpKey=T

gg.handler.kafka.format.metaColumnsTemplate=${optype},${timestampmicro},${currenttimestampmicro},${timestamp}
```

10.2.38.4.7 Using the Length Delimited Value Formatter

The Length Delimited Value (LDV) Formatter is a row-based formatter. It formats database operations from the source trail file into a length delimited value output. Each insert, update, delete, or truncate operation from the source trail is formatted into an individual length delimited message.

With the length delimited, there are no field delimiters. The fields are variable in size based on the data.

By default, the length delimited maps these column value states into the length delimited value output. Column values for an operation from the source trail file can have one of three states:

- Column has a value —The column value is output with the prefix indicator P.
- Column value is NULL —The default output value is N. The output for the case of a NULL column value is configurable.
- Column value is missing - The default output value is M. The output for the case of a missing column value is configurable.
- [Formatting Message Details](#)
- [Sample Formatted Messages](#)
- [LDV Formatter Configuration Properties](#)
- [Additional Considerations](#)

10.2.38.4.7.1 Formatting Message Details

The default format for output of data is the following:

First is the row Length followed by metadata:

```
<ROW LENGTH><PRESENT INDICATOR><FIELD LENGTH><OPERATION TYPE><PRESENT INDICATOR><FIELD LENGTH><FULLY QUALIFIED TABLE NAME><PRESENT INDICATOR><FIELD LENGTH><OPERATION
```

TIMESTAMP><PRESENT INDICATOR><FIELD LENGTH><CURRENT TIMESTAMP><PRESENT INDICATOR><FIELD LENGTH><TRAIL POSITION><PRESENT INDICATOR><FIELD LENGTH><TOKENS>

Or

<ROW LENGTH><FIELD LENGTH><FULLY QUALIFIED TABLE NAME><FIELD LENGTH><OPERATION
TIMESTAMP><FIELD LENGTH><CURRENT TIMESTAMP><FIELD LENGTH><TRAIL POSITION><FIELD
LENGTH><TOKENS>

Next is the row data:

<PRESENT INDICATOR><FIELD LENGTH><COLUMN 1 VALUE><PRESENT INDICATOR><FIELD
LENGTH><COLUMN N VALUE>

10.2.38.4.7.2 Sample Formatted Messages

Insert Message:

0133P01IP161446749136000000P161529311765024000P262015-11-05
18:45:36.000000P04WILLP191994-09-30 15:33:00P03CARP03144P0817520.00P013P03100

Update Message

0133P01UP161446749139000000P161529311765035000P262015-11-05
18:45:39.000000P04BILLP191995-12-31 15:00:00P03CARP03765P0814000.00P013P03100

Delete Message

0136P01DP161446749139000000P161529311765038000P262015-11-05
18:45:39.000000P04DAVEP191993-11-03
07:51:35P05PLANEP03600P09135000.00P012P03200

10.2.38.4.7.3 LDV Formatter Configuration Properties

Table 10-59 LDV Formatter Configuration Properties

Properties	Require d/ Optional	Legal Values	Default	Explanation
gg.handler.name. format.binaryLen gthMode	Optional	true false	false	The output can be controlled to display the field or record length in either binary or ASCII format. If set to true, the record or field length is represented in binary format else in ASCII.
gg.handler.name. format.recordLen gth	Optional	4 8	true	Set to true, the record length is represented using either a 4 or 8-byte big Endian integer. Set to false, the string representation of the record length with padded value with configured length of 4 or 8 is used.
gg.handler.name. format.fieldLeng th	Optional	2 4	true	Set to true, the record length is represented using either a 2 or 4-byte big Endian integer. Set to false, the string representation of the record length with padded value with configured length of 2 or 4 is used.
gg.handler.name. format.format	Optional	true false	true	Use to configure the Pindicator with MetaColumn. Set to false, enables the indicator P before the MetaColumns. If set to true, disables the indicator.

Table 10-59 (Cont.) LDV Formatter Configuration Properties

Properties	Require d/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.format.presentValue</code>	Optional	Any string	P	Use to configure what is included in the output when a column value is present. This value supports CDATA[] wrapping.
<code>gg.handler.name.format.missingValue</code>	Optional	Any string	M	Use to configure what is included in the output when a missing value is present. This value supports CDATA[] wrapping.
<code>gg.handler.name.format.nullValue</code>	Optional	Any string	N	Use to configure what is included in the output when a NULL value is present. This value supports CDATA[] wrapping.
<code>gg.handler.name.format.metaColumnsTemplate</code>	Optional	See Metacolumn Keywords .	None	Use to configure the current meta column information in a simple manner and removes the explicit need of <code>insertOpKey</code>, <code>updateOpKey</code>, <code>deleteOpKey</code>, <code>truncateOpKey</code>, <code>includeTableName</code>, <code>includeOpTimestamp</code>, <code>includeOpType</code>, <code>includePosition</code>, <code>includeCurrentTimestamp</code> and <code>useIso8601Format</code>. A comma-delimited string consisting of one or more templated values represents the template. This example produces a list of meta columns: <pre>{optype}, \${token.ROWID}, \${sys.username}, \${currenttimestamp}</pre> See Metacolumn Keywords.
<code>gg.handler.name.format.pkUpdateHandling</code>	Optional	<code>abend</code> <code>update</code> <code>delete-insert</code>	<code>abend</code>	Specifies how the formatter handles update operations that change a primary key. Primary key operations can be problematic for the text formatter and require special consideration by you. <code>abend</code> : indicates the process will abend <code>update</code> : indicates the process will treat this as a normal update <code>delete-insert</code>: indicates the process handles this as a delete and an insert. Full supplemental logging must be enabled for this to work. Without full before and after row images, the insert data will be incomplete.

Table 10-59 (Cont.) LDV Formatter Configuration Properties

Properties	Require d/ Optional	Legal Values	Default	Explanation
<code>gg.handler.name.format.encoding</code>	Optional	Any encoding name or alias supported by Java.	The native system encoding of the machine hosting the Oracle Golden Gate process.	Use to set the output encoding for character data and columns.

For more information about the Metacolumn keywords, see [Metacolumn Keywords](#). This is an example that would produce a list of metacolumns:

```
${optype}, ${token.ROWID}, ${sys.username}, ${currenttimestamp}
```

Review a Sample Configuration

```
#The LDV Handler
gg.handler.filewriter.format=binary
gg.handler.filewriter.format.binaryLengthMode=false
gg.handler.filewriter.format.recordLength=4
gg.handler.filewriter.format.fieldLength=2
gg.handler.filewriter.format.legacyFormat=false
gg.handler.filewriter.format.presentValue=CDATA[P]
gg.handler.filewriter.format.missingValue=CDATA[M]
gg.handler.filewriter.format.nullValue=CDATA[N]
gg.handler.filewriter.format.metaColumnsTemplate=${optype},${timestampmicro},${currenttimestampmicro},${timestamp}
gg.handler.filewriter.format.pkUpdateHandling=abend
```

10.2.38.4.7.4 Additional Considerations

Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) differs from RDBMSs in how data is stored. Update and delete operations in an RDBMS result in a change to the existing data. Data is not changed in GG for DAA, it is simply appended to existing data. The current state of a given row becomes a consolidation of all of the existing operations for that row in the HDFS system.

Primary Key Updates

Primary key update operations require special consideration and planning for GG for DAA integrations. Primary key updates are update operations that modify one or more of the primary keys for the given row from the source database. Since data is simply appended in GG for DAA, a primary key update operation looks more like a new insert than an update without any special handling. The Length Delimited Value Formatter provides specialized handling for primary keys that is configurable to you. These are the configurable behaviors:

Table 10-60 Primary Key Update Behaviors

Value	Description
Abend	The default behavior is that the length delimited value formatter will abend in the case of a primary key update.
Update	With this configuration the primary key update will be treated just like any other update operation. This configuration alternative should only be selected if you can guarantee that the primary key that is being changed is not being used as the selection criteria when selecting row data from a GG for DAA system.
Delete-Insert	Using this configuration the primary key update is treated as a special case of a delete using the before image data and an insert using the after image data. This configuration may more accurately model the effect of a primary key update in a GG for DAA. However, if this configuration is selected it is important to have full supplemental logging enabled on replication at the source database. Without full supplemental logging, the delete operation will be correct, but the insert operation do not contain all of the data for all of the columns for a full representation of the row data in GG for dAA

Consolidating Data

GG for DAA simply append data to the underlying storage. Typically, analytic tools spawn map reduce programs that traverse the data files and consolidate all the operations for a given row into a single output. It is important to have an indicator of the order of operations. The Length Delimited Value Formatter provides a number of metadata fields to fulfill this need. The operation timestamp may be sufficient to fulfill this requirement. However, two update operations may have the same operation timestamp especially if they share a common transaction. The trail position can provide a tie breaking field on the operation timestamp. Lastly, the current timestamp may provide the best indicator of order of operations in GG for DAA.

10.2.38.4.8 Using the XML Formatter

The XML Formatter formats before-image and after-image data from the source trail file into an XML document representation of the operation data. The format of the XML document is effectively the same as the XML format in the previous releases of the Oracle GoldenGate Java Adapter.

- [Message Formatting Details](#)
- [Sample XML Messages](#)
- [XML Schema](#)
- [XML Formatter Configuration Properties](#)
- [Review a Sample Configuration](#)
- [Metadata Change Events](#)
- [Primary Key Updates](#)

10.2.38.4.8.1 Message Formatting Details

The XML formatted messages contain the following information:

Table 10-61 XML formatting details

Value	Description
table	The fully qualified table name.
type	The operation type.
current_ts	The current timestamp is the time when the formatter processed the current operation record. This timestamp follows the ISO-8601 format and includes micro second precision. Replaying the trail file does not result in the same timestamp for the same operation.
pos	The position from the source trail file.
numCols	The total number of columns in the source table.
col	The col element is a repeating element that contains the before and after images of operation data.
tokens	The tokens element contains the token values from the source trail file.

10.2.38.4.8.2 Sample XML Messages

The following sections provide sample XML messages.

- [Sample Insert Message](#)
- [Sample Update Message](#)
- [Sample Delete Message](#)
- [Sample Truncate Message](#)

10.2.38.4.8.2.1 Sample Insert Message

```
<?xml version='1.0' encoding='UTF-8'?>
<operation table='GG.TCUSTORD' type='I' ts='2013-06-02 22:14:36.000000'
current_ts='2015-10-06T12:21:50.100001' pos='00000000000000001444' numCols='7'>
  <col name='CUST_CODE' index='0'>
    <before missing='true'/>
    <after><![CDATA[WILL]]></after>
  </col>
  <col name='ORDER_DATE' index='1'>
    <before missing='true'/>
    <after><![CDATA[1994-09-30:15:33:00]]></after>
  </col>
  <col name='PRODUCT_CODE' index='2'>
    <before missing='true'/>
    <after><![CDATA[CAR]]></after>
  </col>
  <col name='ORDER_ID' index='3'>
    <before missing='true'/>
    <after><![CDATA[144]]></after>
  </col>
  <col name='PRODUCT_PRICE' index='4'>
    <before missing='true'/>
    <after><![CDATA[17520.00]]></after>
  </col>
  <col name='PRODUCT_AMOUNT' index='5'>
    <before missing='true'/>
    <after><![CDATA[3]]></after>
  </col>
```

```

    <col name='TRANSACTION_ID' index='6'>
      <before missing='true' />
      <after><![CDATA[100]]></after>
    </col>
  <tokens>
    <token>
      <Name><![CDATA[R]]></Name>
      <Value><![CDATA[AADPkVAAEAAEqL2AAA]]></Value>
    </token>
  </tokens>
</operation>

```

10.2.38.4.8.2.2 Sample Update Message

```

<?xml version='1.0' encoding='UTF-8'?>
<operation table='GG.TCUSTORD' type='U' ts='2013-06-02 22:14:41.000000'
current_ts='2015-10-06T12:21:50.413000' pos='00000000000000002891' numCols='7'>
  <col name='CUST_CODE' index='0'>
    <before><![CDATA[BILL]]></before>
    <after><![CDATA[BILL]]></after>
  </col>
  <col name='ORDER_DATE' index='1'>
    <before><![CDATA[1995-12-31:15:00:00]]></before>
    <after><![CDATA[1995-12-31:15:00:00]]></after>
  </col>
  <col name='PRODUCT_CODE' index='2'>
    <before><![CDATA[CAR]]></before>
    <after><![CDATA[CAR]]></after>
  </col>
  <col name='ORDER_ID' index='3'>
    <before><![CDATA[765]]></before>
    <after><![CDATA[765]]></after>
  </col>
  <col name='PRODUCT_PRICE' index='4'>
    <before><![CDATA[15000.00]]></before>
    <after><![CDATA[14000.00]]></after>
  </col>
  <col name='PRODUCT_AMOUNT' index='5'>
    <before><![CDATA[3]]></before>
    <after><![CDATA[3]]></after>
  </col>
  <col name='TRANSACTION_ID' index='6'>
    <before><![CDATA[100]]></before>
    <after><![CDATA[100]]></after>
  </col>
  <tokens>
    <token>
      <Name><![CDATA[R]]></Name>
      <Value><![CDATA[AADPkVAAEAAEqLzAAA]]></Value>
    </token>
  </tokens>
</operation>

```

10.2.38.4.8.2.3 Sample Delete Message

```

<?xml version='1.0' encoding='UTF-8'?>
<operation table='GG.TCUSTORD' type='D' ts='2013-06-02 22:14:41.000000'
current_ts='2015-10-06T12:21:50.415000' pos='00000000000000004338' numCols='7'>
  <col name='CUST_CODE' index='0'>
    <before><![CDATA[DAVE]]></before>
    <after missing='true' />
  </col>
  <col name='ORDER_DATE' index='1'>

```

```

    <before><![CDATA[1993-11-03:07:51:35]]></before>
    <after missing='true' />
  </col>
  <col name='PRODUCT_CODE' index='2'>
    <before><![CDATA[PLANE]]></before>
    <after missing='true' />
  </col>
  <col name='ORDER_ID' index='3'>
    <before><![CDATA[600]]></before>
    <after missing='true' />
  </col>
  <col name='PRODUCT_PRICE' index='4'>
    <missing/>
  </col>
  <col name='PRODUCT_AMOUNT' index='5'>
    <missing/>
  </col>
  <col name='TRANSACTION_ID' index='6'>
    <missing/>
  </col>
  <tokens>
    <token>
      <Name><![CDATA[L]]></Name>
      <Value><![CDATA[206080450]]></Value>
    </token>
    <token>
      <Name><![CDATA[6]]></Name>
      <Value><![CDATA[9.0.80330]]></Value>
    </token>
    <token>
      <Name><![CDATA[R]]></Name>
      <Value><![CDATA[AADPKvAAEAAEqLzAAC]]></Value>
    </token>
  </tokens>
</operation>

```

10.2.38.4.8.2.4 Sample Truncate Message

```

<?xml version='1.0' encoding='UTF-8'?>
<operation table='GG.TCUSTORD' type='T' ts='2013-06-02 22:14:41.000000'
current_ts='2015-10-06T12:21:50.415001' pos='00000000000000004515' numCols='7'>
  <col name='CUST_CODE' index='0'>
    <missing/>
  </col>
  <col name='ORDER_DATE' index='1'>
    <missing/>
  </col>
  <col name='PRODUCT_CODE' index='2'>
    <missing/>
  </col>
  <col name='ORDER_ID' index='3'>
    <missing/>
  </col>
  <col name='PRODUCT_PRICE' index='4'>
    <missing/>
  </col>
  <col name='PRODUCT_AMOUNT' index='5'>
    <missing/>
  </col>
  <col name='TRANSACTION_ID' index='6'>
    <missing/>
  </col>
  <tokens>

```

```

    <token>
      <Name><![CDATA[R]]></Name>
      <Value><![CDATA[AADPkVAAEAAEqL2AAB]]></Value>
    </token>
  </tokens>
</operation>

```

10.2.38.4.8.3 XML Schema

The XML Formatter does not generate an XML schema (XSD). The XSD applies to all messages generated by the XML Formatter. The following XSD defines the structure of the XML documents that are generated by the XML Formatter.

```

<xs:schema attributeFormDefault="unqualified"
  elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="operation">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="col" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="before" minOccurs="0">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:string" name="missing"
use="optional"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <xs:element name="after" minOccurs="0">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute type="xs:string" name="missing"
use="optional"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <xs:element type="xs:string" name="missing" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute type="xs:string" name="name"/>
            <xs:attribute type="xs:short" name="index"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="tokens" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="token" maxOccurs="unbounded" minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="Name"/>
                    <xs:element type="xs:string" name="Value"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

```

    <xs:attribute type="xs:string" name="table"/>
    <xs:attribute type="xs:string" name="type"/>
    <xs:attribute type="xs:string" name="ts"/>
    <xs:attribute type="xs:dateTime" name="current_ts"/>
    <xs:attribute type="xs:long" name="pos"/>
    <xs:attribute type="xs:short" name="numCols"/>
  </xs:complexType>
</xs:element>
</xs:schema>

```

10.2.38.4.8.4 XML Formatter Configuration Properties

Table 10-62 XML Formatter Configuration Properties

Properties	Optional Y/N	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.format.insert0</code> <code>pKey</code>	Optional	Any string	I	Indicator to be inserted into the output record to indicate an insert operation.
<code>gg.handler.name</code> <code>.format.update0</code> <code>pKey</code>	Optional	Any string	U	Indicator to be inserted into the output record to indicate an update operation.
<code>gg.handler.name</code> <code>.format.delete0</code> <code>pKey</code>	Optional	Any string	D	Indicator to be inserted into the output record to indicate a delete operation.
<code>gg.handler.name</code> <code>.format.truncat</code> <code>eOpKey</code>	Optional	Any string	T	Indicator to be inserted into the output record to indicate a truncate operation.
<code>gg.handler.name</code> <code>.format.encoding</code>	Optional	Any legal encoding name or alias supported by Java.	UTF-8 (the XML default)	The output encoding of generated XML documents.
<code>gg.handler.name</code> <code>.format.include</code> Prolog	Optional	true false	false	Determines whether an XML prolog is included in generated XML documents. An XML prolog is optional for well-formed XML. An XML prolog resembles the following: <code><?xml version='1.0' encoding='UTF-8' ?></code>
<code>gg.handler.name</code> <code>.format.iso8601</code> Format	Optional	true false	true	Controls the format of the current timestamp in the XML message. The default adds a T between the date and time. Set to false to suppress the T between the date and time and instead include blank space.
<code>gg.handler.name</code> <code>.format.missing</code>	Optional	true false	true	Set to true, the XML output displays the missing column value of the before and after image.
<code>gg.handler.name</code> <code>.format.missing</code> After	Optional	true false	true	Set to true, the XML output displays the missing column value of the after image.

Table 10-62 (Cont.) XML Formatter Configuration Properties

Properties	Optional Y/N	Legal Values	Default	Explanation
<code>gg.handler.name</code> <code>.format.missing</code> Before	Optional	true false	true	Set to true, the XML output displays the missing column value of the before image.
<code>gg.handler.name</code> <code>.format.metaCol</code> umnsTemplate	Optional	See Metacolumn Keywords .	None	The current meta column information can be configured in a simple manner and removes the explicit need to use: <code>insertOpKey</code> <code>updateOpKey</code> <code>deleteOpKey</code> <code>truncateOpKey</code> <code>includeTableName</code> <code>includeOpTimestamp</code> <code>includeOpType</code> <code>includePosition</code> <code>includeCurrentTimestamp</code>, <code>useIso8601Format</code> It is a comma-delimited string consisting of one or more templated values that represent the template. For more information about the Metacolumn keywords, see Metacolumn Keywords.

10.2.38.4.8.5 Review a Sample Configuration

The following is a sample configuration for the XML Formatter in the Java Adapter properties file:

```
gg.handler.hdfs.format=xml
gg.handler.hdfs.format.insertOpKey=I
gg.handler.hdfs.format.updateOpKey=U
gg.handler.hdfs.format.deleteOpKey=D
gg.handler.hdfs.format.truncateOpKey=T
gg.handler.hdfs.format.encoding=ISO-8859-1
gg.handler.hdfs.format.includeProlog=false
```

10.2.38.4.8.6 Metadata Change Events

The XML Formatter seamlessly handles metadata change events. A metadata change event does not result in a change to the XML schema. The XML schema is designed to be generic so that the same schema represents the data of any operation from any table.

If the replicated database and upstream Oracle GoldenGate replication process can propagate metadata change events, the XML Formatter can take action when metadata changes. Changes in the metadata are reflected in messages after the change. For example, when a column is added, the new column data appears in XML messages for the table.

10.2.38.4.8.7 Primary Key Updates

Updates to a primary key require no special handling by the XML formatter. The XML formatter creates messages that model database operations. For update operations, this includes before and after images of column values. Primary key changes are represented in this format as a change to a column value just like a change to any other column value.

10.2.38.5 Stage and Merge Data Warehouse Replication

Data warehouse targets typically support Massively Parallel Processing (MPP). The cost of a single Data Manipulation Language (DML) operation is comparable to the cost of execution of batch DMLs.

Therefore, for better throughput the change data from the Oracle GoldenGate trails can be staged in micro batches at a temporary staging location, and the staged data records are merged into the data warehouse target table using the respective data warehouse's merge SQL statement. This section outlines an approach to replicate change data records from source databases to target data warehouses using stage and merge.

This chapter contains:

- [Steps for Stage and Merge](#)
- [Hive Stage and Merge](#)
Hive is a data warehouse infrastructure built on top of Hadoop. It provides tools to enable easy data ETL, a mechanism to put structures on the data, and the capability for querying and analysis of large data sets stored in Hadoop files.

10.2.38.5.1 Steps for Stage and Merge

- [Stage](#)
In this step the change data records in the Oracle GoldenGate trail files are pushed into a staging location. The staging location is typically a cloud object store such as OCI, AWS S3, Azure Data Lake, or Google Cloud Storage.
- [Merge](#)
In this step the change data files in the object store are viewed as an external table defined in the data warehouse. The data in the external staging table is merged onto the target table.
- [Configuration of Handlers](#)
File Writer(FW) handler needs to be configured to generate local staging files that contain change data from the Oracle GoldenGate trail files.
- [File Writer Handler](#)
File Writer (FW) handler is typically configured to generate files partitioned by table using the configuration `gg.handler.{name}.partitionByTable=true`.
- [Operation Aggregation](#)
Operation aggregation is the process of aggregating (merging/compressing) multiple operations on the same row into a single output operation based on a threshold.
- [Object Store Event handler](#)
The File Writer handler needs to be chained with an object store Event handler. Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) supports uploading files to most cloud object stores such as OCI, AWS S3, and Azure Data Lake.
- [JDBC Metadata Provider](#)
If the data warehouse supports JDBC connection, then the JDBC metadata provider needs to be enabled.
- [Stage and Merge Sample Configuration](#)
A working configuration for the respective data warehouse is available under the directory `AdapterExamples/big-data/data-warehouse-utils/<target>/`.

- [Variables in the Merge Script](#)
Typically, variables appear at the beginning of the Oracle provided script. There are lines starting with #TODO: that document the changes required for variables in the script.
- [SQL Statements in the Merge Script](#)
The SQL statements in the shell script needs to be customized. There are lines starting with #TODO: that document the changes required for SQL statements.
- [Merge Script Functions](#)
- [Prerequisites](#)
- [Limitations](#)

10.2.38.5.1.1 Stage

In this step the change data records in the Oracle GoldenGate trail files are pushed into a staging location. The staging location is typically a cloud object store such as OCI, AWS S3, Azure Data Lake, or Google Cloud Storage.

This can be achieved using File Writer handler and one of the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) for object store Event handlers.

10.2.38.5.1.2 Merge

In this step the change data files in the object store are viewed as an external table defined in the data warehouse. The data in the external staging table is merged onto the target table.

Merge SQL uses the external table as the staging table. The merge is a batch operation leading to better throughput.

10.2.38.5.1.3 Configuration of Handlers

File Writer(FW) handler needs to be configured to generate local staging files that contain change data from the Oracle GoldenGate trail files.

The FW handler needs to be chained to an object store Event handler that can upload the staging files into a staging location.

The staging location is typically a cloud object store, such as AWS S3 or Azure Data Lake.

10.2.38.5.1.4 File Writer Handler

File Writer (FW) handler is typically configured to generate files partitioned by table using the configuration `gg.handler.{name}.partitionByTable=true`.

In most cases FW handler is configured to use the Avro Object Container Format (OCF) formatter.

The output file format could change based on the specific data warehouse target.

10.2.38.5.1.5 Operation Aggregation

Operation aggregation is the process of aggregating (merging/compressing) multiple operations on the same row into a single output operation based on a threshold.

Operation Aggregation needs to be enabled for stage and merge replication using the configuration `gg.aggregate.operations=true`.

10.2.38.5.1.6 Object Store Event handler

The File Writer handler needs to be chained with an object store Event handler. Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) supports uploading files to most cloud object stores such as OCI, AWS S3, and Azure Data Lake.

10.2.38.5.1.7 JDBC Metadata Provider

If the data warehouse supports JDBC connection, then the JDBC metadata provider needs to be enabled.

10.2.38.5.1.8 Stage and Merge Sample Configuration

A working configuration for the respective data warehouse is available under the directory `AdapterExamples/big-data/data-warehouse-utils/<target>/`.

This directory contains the following:

- replicat parameter (.prm) file.
- replicat properties file that contains the FW handler and all the Event handler configuration.
- DDL file for the sample table used in the merge script.
- Merge script for the specific data warehouse. This script contains SQL statements tested using the sample table defined in the DDL file.

10.2.38.5.1.9 Variables in the Merge Script

Typically, variables appear at the beginning of the Oracle provided script. There are lines starting with #TODO: that document the changes required for variables in the script.

Example:

```
#TODO: Edit this. Provide the replicat group name.
repName=RBD

#TODO: Edit this. Ensure each replicat uses a unique prefix.
stagingTablePrefix=${repName}_STAGE_

#TODO: Edit the AWS S3 bucket name.
bucket=<AWS S3 bucket name>

#TODO: Edit this variable as needed.
s3Location="'s3://${bucket}/${dir}/'"

#TODO: Edit AWS credentials awsKeyId and awsSecretKey
awsKeyId=<AWS Access Key Id>
awsSecretKey=<AWS Secret key>
```

The variables repName and stagingTablePrefix are relevant for all the data warehouse targets.

10.2.38.5.1.10 SQL Statements in the Merge Script

The SQL statements in the shell script needs to be customized. There are lines starting with #TODO: that document the changes required for SQL statements.

In most cases, we need to double quote " identifiers in the SQL statement. The double quote needs to be escaped in the script using backslash. For example: \".

Oracle provides a working example of SQL statements for a single table with a pre-defined set of columns defined in the sample DDL file. You need to add new sections for your own tables as part of if-else code block in the script.

Example:

```
if [ "${tableName}" == "DBO.TCUSTORD" ]
then
    #TODO: Edit all the column names of the staging and target tables.
    # The merge SQL example here is configured for the example table defined in the DDL
    file.
    # Oracle provided SQL statements

    # TODO: Add similar SQL queries for each table.
elif [ "${tableName}" == "DBO.ANOTHER_TABLE" ]
then

    #Edit SQLs for this table.
fi
```

10.2.38.5.1.11 Merge Script Functions

The script is coded to include the following shell functions:

- main
- validateParams
- process
- processTruncate
- processDML
- dropExternalTable
- createExternalTable
- merge

The script has code comments for you to infer the purpose of each function.

Merge Script main function

The function main is the entry point of the script. The processing of the staged changed data file begin here.

This function invokes two functions: validateParams and process.

The input parameters to the script is validated in the function: validateParams.

Processing resumes in the process function if validation is successful.

Merge Script process function

This function processes the operation records in the staged change data file and invokes processTruncate or processDML as needed.

Truncate operation records are handled in the function processTruncate. Insert, Update, and Delete operation records are handled in the function processDML.

Merge Script merge function

The merge function invoked by the function processDML contains the merge SQL statement that will be executed for each table.

The key columns to be used in the merge SQL's ON clause needs to be customized.

To handle key columns with null values, the ON clause uses data warehouse specific NVL functions. Example for a single key column "C01Key":

```
ON ((NVL(CAST(TARGET.\"C01Key\" AS VARCHAR(4000)), '{uuid}')=NVL(CAST(STAGE.\"C01Key\" AS VARCHAR(4000)), '{uuid}'))`
```

The column names in the merge statement's update and insert clauses also needs to be customized for every table.

Merge Script createExternalTable function

The createExternalTable function invoked by the function processDML creates an external table that is backed by the file in the respective object store file.

In this function, the DDL SQL statement for the external table should be customized for every target table to include all the target table columns.

In addition to the target table columns, the external table definition also consists of three meta-columns: optype, position, and fieldmask.

The data type of the meta-columns should not be modified. The position of the meta-columns should not be modified in the DDL statement.

10.2.38.5.1.12 Prerequisites

- The Command handler merge scripts are available, starting from Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) release 19.1.0.0.8.
- The respective data warehouse's command line programs to execute SQL queries must be installed on the machine where GG for DAA is installed.

10.2.38.5.1.13 Limitations

Primary key update operations are split into delete and insert pair. In case the Oracle GoldenGate trail file doesn't contain column values for all the columns in the respective table, then the missing columns gets updated to null on the target table.

10.2.38.5.2 Hive Stage and Merge

Hive is a data warehouse infrastructure built on top of Hadoop. It provides tools to enable easy data ETL, a mechanism to put structures on the data, and the capability for querying and analysis of large data sets stored in Hadoop files.

- [Data Flow](#)
- [Configuration](#)
The directory AdapterExamples/big-data/data-warehouse-utils/hive/ in the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) install contains all the configuration and scripts needed needed for replication to Hive using stage and merge.
- [Merge Script Variables](#)
- [Prerequisites](#)

10.2.38.5.2.1 Data Flow

- File Writer (FW) handler is configured to generate files in Avro Object Container Format (OCF).
- The HDFS Event handler is used to push the Avro OCF files into Hadoop.

10.2.38.5.2.2 Configuration

The directory `AdapterExamples/big-data/data-warehouse-utils/hive/` in the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) install contains all the configuration and scripts needed for replication to Hive using stage and merge.

The following are the files:

- `hive.prm`: The replicat parameter file.
- `hive.props`: The replicat properties file that stages data to Hadoop.
- `hive.sh`: The bash-shell script that reads data staged in Hadoop and merges data to Hive target table.
- `hive-ddl.sql`: The DDL statement that contains sample target table used in the script `hive.sh`.

Edit the properties indicated by the `#TODO`: comments in the properties file `hive.props`.

The bash-shell script function `merge()` contains SQL statements that needs to be customized for your target tables.

10.2.38.5.2.3 Merge Script Variables

Modify the variables needs as needed:

```
#TODO: Modify the location of the OGGBD dirdef directory where the Avro schema files exist.
```

```
avroSchemaDir=/opt/ogg/dirdef
```

```
#TODO: Edit the JDBC URL to connect to hive.
```

```
hiveJdbcUrl=jdbc:hive2://localhost:10000/default
```

```
#TODO: Edit the JDBC user to connect to hive.
```

```
hiveJdbcUser=APP
```

```
#TODO: Edit the JDBC password to connect to hive.
```

```
hiveJdbcPassword=mine
```

```
#TODO: Edit the replicat group name.
```

```
repName=HIVE
```

```
#TODO: Edit this. Ensure each replicat uses a unique prefix.
```

```
stagingTablePrefix=${repName}_STAGE_
```

10.2.38.5.2.4 Prerequisites

The following are the prerequisites:

- The merge script `hive.sh` requires command line program `beeLine` to be installed on the machine where Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) replicat is installed.
- The custom script `hive.sh` uses the merge SQL statement.
Hive Query Language (Hive QL) introduced support for merge in Hive version 2.2.

10.2.38.6 Template Keywords

The templating functionality allows you to use a mix of constants and/or keywords for context based resolution of string values at runtime. The templating functionality is used extensively in the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) to resolve file paths, file names, topic names, or message keys. This appendix describes the keywords and their associated arguments if applicable. Additionally, there are examples showing templates and resolved values.

Template Keywords

This table includes a column if the keyword is supported for transaction level messages.

Keyword	Explanation	Transaction Message Support
<code>\${fullyQualifiedTableName}</code>	Resolves to the fully qualified table name including the period (.) delimiter between the catalog, schema, and table names. For example, TEST.DBO.TABLE1.	No
<code>\${catalogName}</code>	Resolves to the catalog name.	No
<code>\${schemaName}</code>	Resolves to the schema name.	No
<code>\${tableName}</code>	Resolves to the short table name.	No
<code>\${opType}</code>	Resolves to the type of the operation: (INSERT, UPDATE, DELETE, or TRUNCATE)	No
<code>\${primaryKeys[]}</code>	The first parameter is optional and allows you to set the delimiter between primary key values. The default is <code>_</code> .	No
<code>\${position}</code>	The sequence number of the source trail file followed by the offset (RBA).	Yes
<code>\${opTimestamp}</code>	The operation timestamp from the source trail file.	Yes
<code>\${emptyString}</code>	Resolves to <code>""</code> .	Yes
<code>\${groupName}</code>	Resolves to the name of the Replicat process. If using coordinated delivery, it resolves to the name of the Replicat process with the Replicate thread number appended.	Yes

Keyword	Explanation	Transaction Message Support
<code>\${staticMap[]}</code> or <code>\${staticMap[][]}</code>	Resolves to a static value where the key is the fully-qualified table name. The keys and values are designated inside of the square brace in the following format: <code>\${staticMap[DBO.TABLE1=value1,DBO.TABLE2=value2]}</code> The second parameter is an optional default value. If the value cannot be located using the lookup by the table name, then the default value will be used instead.	No
<code>\${xid}</code>	Resolves the transaction id.	Yes
<code>\${columnValue[][]}</code> or <code>\${columnValue[][][]}</code>	Resolves to a column value where the key is the fully-qualified table name and the value is the column name to be resolved. For example: <code>\$</code> <code>{columnValue[DBO.TABLE1=COL1,DBO.TABLE2=COL2]}</code> The second parameter is optional and allows you to set the value to use if the column value is null. The default is an empty string <code>""</code>. The third parameter is optional and allows you to set the value to use if the column value is missing. The default is an empty string <code>""</code>. If the <code>\${columnValue}</code> keyword is used in partitioning, then only the column name needs to be set. Only the HDFS Handler and the File Writer Handler support partitioning. In the case of partitioning, the table name is already known because partitioning configuration is separate for each and every source table. The following is an example of <code>\${columnValue}</code> when used in the context of partitioning: <code>\${columnValue[COL1]}</code> or <code>\${columnValue[COL2][NULL][MISSING]}</code>	No

Keyword	Explanation	Transaction Message Support
<code>\${currentTimestamp}</code> Or <code>\${currentTimestamp[]}</code>	Resolves to the current timestamp. You can control the format of the current timestamp using the Java based formatting as described in the SimpleDateFormat class, see https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html Examples: <code>\${currentTimestamp}\$</code> <code>{currentTimestamp[yyyy-MM-dd HH:mm:ss.SSS]}</code>	Yes
<code>\${null}</code>	Resolves to a NULL string.	Yes
<code>\${custom[]}</code>	It is possible to write a custom value resolver. If required, contact Oracle Support.	Implementation dependent
<code>\${token[]}</code>	Resolves a token value.	No
<code>\${toLowerCase[]}</code>	Keyword to convert to argument to lower case. Argument can be constants, keywords, or combination of both.	Yes
<code>\${toUpperCase[]}</code>	Keyword to convert to argument to upper case. Argument can be constants, keywords, or combination of both.	Yes

Keyword	Explanation	Transaction Message Support
<code>\${substring[][]}</code> Or <code>\${substring[][][]}</code>	Keyword to perform a substring operation on the configured content. 1. The string on which the substring functionality is acting. Can be nested keywords, constants, or a combination of both. 2. The starting index. 3. The ending index. (If not provided then the end of the input string.) \$ {substring[thisisfun] [4]} returns isfun. \$ {substring[thisisfun] [4][6]} returns is.	Yes

 **Not
e**

Performing a substring function means that an array index out of bounds condition can occur at runtime. This occurs if the configured starting index or ending

Keyword	Explanation	Transaction Message Support
	g index is beyond the length of the string currently being acted upon. The \${sub string} function does not throw a runtime exception. It instead detects an array index out of bounds condition and in that case does not execute the substring function.	

Keyword	Explanation	Transaction Message Support
<code>\${regex[][][]}</code>	Keyword to apply a regular expressions to search and replace content. This has three required parameters: 1. The string on which the regular expression search and replace functionality is acting. Can be nested keywords or constants or a combination. 2. The regular expression search string. 3. The regular expression replacement string.	Yes
<code>\${operationCount}</code>	Keyword to resolve the count of operations.	Yes
<code>\${insertCount}</code>	Keyword to resolve the count of insert operations.	Yes
<code>\${deleteCount}</code>	Keyword to resolve the count of delete operations.	Yes
<code>\${updateCount}</code>	Keyword to resolve the count of update operations.	Yes
<code>\${truncateCount}</code>	Keyword to resolve the count of truncate operations.	Yes
<code>\${uuid}</code>	Keyword to resolve a universally unique identifier (UUID). This is a 36 character string guaranteed to be unique. An example UUID: 7f6e4529-e387-48c1-a1b6-3e7a4146b211	Yes

Example Templates

The following describes example template configuration values and the resolved values.

Example Template	Resolved Value
<code>\${groupName}_\${fullyQualifiedTableName}</code>	KAFKA001_DB0.TABLE1
<code>prefix_\${schemaName}_\${tableName}_suffix</code>	prefix_DB0_TABLE1_suffix
<code>\${currentTimestamp[yyyy-MM-dd HH:mm:ss.SSS]}</code>	2017-05-17 11:45:34.254
A_STATIC_VALUE	A_STATIC_VALUE

10.2.38.7 Velocity Dependencies

Starting Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) release 21.1.0.0.0, the Velocity jar files have been removed from the packaging.

For the Velocity formatting to work, you need to download the jars and include them in their runtime by modifying the `gg.classpath`.

The maven coordinates for Velocity are as follows:

Maven groupId: org.apache.velocity

Maven artifactId: velocity

Version: 1.7

10.2.39 Oracle GoldenGate Java Delivery

This part of the book contains information on using Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) to process transaction records and apply it to various targets by means of Java module.

For more information, see [Understanding Oracle GoldenGate for Distributed Applications and Analytics](#).

- [Configuring Java Delivery](#)
- [Running Java Delivery](#)
- [Configuring Event Handlers](#)
- [Java Delivery Properties](#)
- [Developing Custom Filters, Formatters, and Handlers](#)
- [Configuring Data Transforms](#)

10.2.39.1 Configuring Java Delivery

- [Configuring the JRE in the Properties File](#)
- [Configuring Oracle GoldenGate for Java Delivery](#)
- [Configuring the Java Handlers](#)

10.2.39.1.1 Configuring the JRE in the Properties File

The current release of Oracle GoldenGate Java Delivery requires Java 8. Refer to the section on configuring Java for how to correctly access Java and the required Java shared libraries. Modify the Adapter Properties file to point to the location of the Oracle GoldenGate for Java main JAR (ggjava.jar) and set any additional JVM runtime boot options as required (these are passed directly to the JVM at startup):

```
jvm.bootoptions=-Djava.class.path=.:ggjava/ggjava.jar -Xmx512m -Xmx64m
```

Note the following options in particular:

- `java.class.path` must include pathing to the core application (ggjava/ggjava.jar). The current directory (.) should be included as well in the classpath. Logging initializes when the JVM is loaded therefore the `java.class.path` variable should include any pathing to logging properties files (such as log4j properties files). The dependency JARs required for logging functionality are included in ggjava.jar and do not need to be explicitly included. Pathing can reference files and directories relative to the Oracle GoldenGate install directory, to allow storing Java property files, Velocity templates and other classpath resources in the `dirprm` subdirectory. It is also possible to append to the classpath in the Java application properties file. Pathing to handler dependency JARs can be added here as well. However, it is considered to be a better practice to use the `gg.classpath` variable to include any handler dependencies.

- The `jvm.bootoptions` property also allows you to control the initial heap size of the JVM (Xms) and the maximum heap size of the JVM (Xmx). Increasing the maximum heap size can improve performance by requiring less frequent garbage collections. Additionally, you may need to increase the maximum heap size if a Java out of memory exception occurs.

Once the properties file is correctly configured for your system, it usually remains unchanged. See [Common Properties](#), for additional configuration options.

10.2.39.1.2 Configuring Oracle GoldenGate for Java Delivery

Java Delivery is compatible with the Oracle GoldenGate Replicat process. Transaction data is read from the Oracle GoldenGate trail files and delivered to the Oracle GoldenGate Java Delivery module across JNI interface. The data is transferred to the Oracle GoldenGate Java Delivery module using the JNI interface. The Java Delivery module is configurable to allow data to be streamed into various targets. The supported targets for the Oracle GoldenGate Java Adapter product include JMS, file writing, and custom integrations. The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) includes all of those integrations and streaming capabilities to its targets.

- [Configuring a Replicat for Java Delivery](#)

10.2.39.1.2.1 Configuring a Replicat for Java Delivery

The Oracle GoldenGate Replicat process can be configured to send transaction data to the Oracle GoldenGate for Java module. Replicat consumes a local trail (for example `dirdat/aa`) and sends the data to the Java Delivery module. The Java module is responsible for processing all the data and applying it to the desired target.

Following is an example of adding a Replicat process:

```
ADD REPLICAT javarep, EXTTRAIL ./dirdat/aa
```

The process names and trail names used in the preceding example can be replaced with any valid name. Process names must be 8 characters or less, trail names must be two characters. In the Replicat parameter file (`javarep.prm`), specify the location of the user exit library.

The Replicat process has transaction grouping built into the application. Transaction grouping can significantly improve performance when streaming data to a target database. Transaction grouping can also significantly improve performance when streaming data to Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA). The Replicat parameter to control transaction grouping is the `GROUPTRANSOPS` variable in the Replicat configuration file. The default value of this variable is 1000 which means the Replicat process will attempt to group 1000 operations into single target transaction. Performance testing has generally shown that the higher the `GROUPTRANSOPS` the better the performance when streaming data to GG for DAA. Setting the `GROUPTRANSOPS` variable to 1 means that the original transaction boundaries from the source trail file (source database) will be maintained.

Table 10-63 User Exit Replicat Parameters

Parameter	Explanation
REPLICAT javarep	All Replicat parameter files start with the Replicat name

Table 10-63 (Cont.) User Exit Replicat Parameters

Parameter	Explanation
SOURCEDEFS ./dirdef/tcust.def	(Optional) If the input trail files do not contain the metadata records, the Replicat process requires metadata describing the trail data. This can come from a database or a source definitions file. This metadata defines the column names and data types in the trail being read (./dirdat/aa).
TARGETDB LIBFILE libggjava.so SET properties= dirprm/javarep.properties	The TARGETDB LIBFILE libggjava.so parameter serves as a trigger to initialize the Java module. The SET clause to specify the Java properties file is optional. If specified, it should contain an absolute or relative path (relative to the Replicat executable) to the properties file for the Java module. The default value is <i>replicat_name.properties</i> in the <i>dirprm</i> directory.
MAP schema.*, TARGET *.*;	The tables to pass to the Java module; tables not included will be skipped. If mapping from source to target tables is required, you can use the <i>MAP source_specification TARGET target_specification</i> .
GROUPTRANSOPS 1000	Group source transactions into a single larger target transaction for improved performance. GROUPTRANSOPS of 1000 is the default setting. GROUPTRANSOPS sets a minimum value rather than an absolute value, to avoid splitting apart source transactions. Replicat waits until it receives all operations from the last source transaction in the group before applying the target transaction. For example, if transaction 1 contains 200 operations, and transaction 2 contains 400 operations, and transaction 3 contains 500 operations, then Replicat transaction contains all 1,100 operations even though GROUPTRANSOPS is set to the default of 1,000. Conversely, Replicat might apply a transaction before reaching the value set by GROUPTRANSOPS if there is no more data in the trail to process.

10.2.39.1.3 Configuring the Java Handlers

The Handlers are integrations with target applications which plug into the Oracle GoldenGate Java Delivery module. It is the Java Handlers which provide the functionality to push data to integration targets such as JMS or Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA). The Java Adapter properties file is used to configure Java Delivery and Java handlers. To test the configuration, users may use the built-in file handler. Here are some example properties, followed by explanations of the properties (comment lines start with #):

```
# the list of active handlers
gg.handlerlist=myhandler
# set properties on 'myhandler'
gg.handler.myhandler.type=file
gg.handler.myhandler.format=tx2xml.vm
gg.handler.myhandler.file=output.xml
```

This property file declares the following:

- Active event handlers. In the example a single event handler is active, called `myhandler`. Multiple handlers may be specified, separated by commas. For example:
`gg.handlerlist=myhandler, yourhandler`

Note

Starting Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) 23ai release, you will be able to specify only a single handler.

- Configuration of the handlers. In the example `myhandler` is declared to be a file type of handler: `gg.handler.myhandler.type=file`

Note

See the documentation for each type of handler (for example, the JMS handler or the file writer handler) for the list of valid properties that can be set.

- The format of the output is defined by the Velocity template `tx2xml.vm`. You may specify your own custom template to define the message format; just specify the path to your template relative to the Java classpath.

This property file is actually a complete example that will write captured transactions to the output file `output.xml`. Other handler types can be specified using the keywords: `jms_text` (or `jms`), `jms_map`, `singlefile` (a file that does not roll), and others. Custom handlers can be implemented, in which case the type would be the fully qualified name of the Java class for the handler. GG for DAA package also contains built in the DAA target types.

Note

See the documentation for each type of handler (for example, the JMS handler or the file writer handler) for the list of valid properties that can be set.

10.2.39.2 Running Java Delivery

- [Starting the Application](#)
- [Restarting the Java Delivery](#)

10.2.39.2.1 Starting the Application

To run the Java Delivery and execute the Java application, you only need an existing Oracle GoldenGate trail file. If the trail file does not contain metadata records, a source definitions file is also required to describe the schema for operations in the trail file. For the examples that follow, a simple TCUSTMER and TCUSTORD trail is used (matching the demo SQL provided with the Oracle GoldenGate software download).

- [Starting Using Replicat](#)

10.2.39.2.1.1 Starting Using Replicat

To run Java Delivery using Replicat, simply start the Replicat process from GGSCI:

```
GGSCI> START REPLICAT javarep
GGSCI> INFO REPLICAT javarep
```

The INFO command returns information similar to the following:

```
REPLICAT JAVAREP          Last Started 2015-09-10 17:25 Status RUNNING
Checkpoint Lag            00:00:00 (updated 00:00:00 ago)
Log Read Checkpoint File  ./dirdat/aa0000002015-09-10 17:50:41.000000
                          RBA 2702
```

10.2.39.2.2 Restarting the Java Delivery

There are two possible checkpoint files when running with Replicat, the Replicat process checkpoint file and the Java Delivery checkpoint file. Both files are located in the `dirchk` directory and created using the following naming conventions.

Replicat checkpoint file

`group_name.cpr`

Java delivery checkpoint file:

`group_name.cpj`

To suppress the creation and use of the Java Delivery checkpoint the Replicat process should be created using the following syntax:

```
ADD REPLICAT myrep EXTTRAIL ./dirdat/tr NODBCHECKPOINT
```

It is the `NODBCHECKPOINT` syntax that disables the creation and use of the Java Delivery checkpoint file.

- [Restarting Java Delivery in Replicat](#)

10.2.39.2.2.1 Restarting Java Delivery in Replicat

The checkpoint handling in Replicat is more straightforward as it includes logic to pick which one out of the two checkpoint information is of higher priority. The logic is as follows:

- If the Java Delivery is started after user manually performed an `ADD` or `ALTER REPLICAT`, then the checkpoint information held by Replicat process will be used as the starting position.
- If the Java Delivery is started without prior manual intervention to alter checkpoint (for example, upon graceful stop or an `abend`), then the checkpoint information held by Java module will be used as the starting position.

For example, restarting a Java Delivery using Replicat at the beginning of a trail looks like the following:

1. Reset the Replicat to the beginning of the trail data:

```
GGSCI> ALTER REPLICAT JAVAREP, EXTSEQNO 0, EXTRBA 0
```

2. Reset the Replicat

```
GGSCI> START JAVAREP
GGSCI> INFO JAVAREP
REPLICAT JAVAREP          Last Started 2015-09-10 17:25 Status RUNNING
Checkpoint Lag            00:00:00 (updated 00:00:00 ago)
Log Read Checkpoint File  ./dirdat/aa000000
2015-09-10 17:50:41.000000 RBA 2702
```

It may take a few seconds for the Replicat process status to report itself as running. Check the report file to see if it abended or is still in the process of starting:

```
GGSCI> VIEW REPORT JAVAREP
```

In the case where the Java Delivery is restarted after a crash or an abend, the last position kept by the Java module will be used when the application restarts.

10.2.39.3 Configuring Event Handlers

This chapter discusses types of event handlers explaining how to specify the event handler to use and what your options are. It explains how to format the output and what you can expect from the Oracle GoldenGate Report file.

- [Specifying Event Handlers](#)
- [JMS Handler](#)
- [File Handler](#)
- [Custom Handlers](#)
- [Formatting the Output](#)
- [Reporting](#)

10.2.39.3.1 Specifying Event Handlers

Processing transaction, operation and metadata events in Java works as follows:

- The Oracle GoldenGate Replicat or Extract process reads local trail data and passes the transactions, operations and database metadata to the Java Delivery Module. Metadata can come from the trail itself, a source definitions file.
- Events are fired by the Java framework, optionally filtered by custom Event Filters.
- Handlers (event listeners) process these events, and process the transactions, operations and metadata. Custom formatters may be applied for certain types of targets.

There are several existing handlers:

- Various built in Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) handlers to apply records to supported GG for DAA targets, see [Replicate Data](#) to configure various handlers supported in GG for DAA.
- JMS message handlers to send to a JMS provider using either a MapMessage, or using a TextMessage with customized formatters.
- A specialized message handler to send JMS messages to Oracle Advanced Queuing (AQ).
- A file writer handler, for writing to a single file, or a rolling file.

Note

The file writer handler is particularly useful as development utility, since the file writer can take the exact same formatter as the JMS TextMessage handler. Using the file writer provides a simple way to test and tune the formatters for JMS without actually sending the messages to JMS

Event handlers can be configured using the main Java property file or they may optionally read in their own properties directly from yet another property file (depending on the handler implementation). Handler properties are set using the following syntax:

```
gg.handler.{name}.someproperty=somevalue
```

This will cause the property *someproperty* to be set to the value *somevalue* for the handler instance identified in the property file by *name*. This *name* is used in the property file to define active handlers and set their properties; it is user-defined.

Implementation note (for Java developers): Following the preceding example: when the handler is instantiated, the method `void setSomeProperty(String value)` will be called on the handler instance, passing in the String value *somevalue*. A JavaBean `PropertyEditor` may also be defined for the handler, in which case the string can be automatically converted to the appropriate type for the setter method. For example, in the Java application properties file, we may have the following:

```
# the list of active handlers: only two are active
gg.handlerlist=one, two
# set properties on 'one'
gg.handler.one.type=file
gg.handler.one.format=com.mycompany.MyFormatter
gg.handler.one.file=output.xml
# properties for handler 'two'
gg.handler.two.type=jms_text
gg.handler.two.format=com.mycompany.MyFormatter
gg.handler.two.properties=jboss.properties
# set properties for handler 'foo'; this handler is ignored
gg.handler.foo.type=com.mycompany.MyHandler
gg.handler.foo.someproperty=somevalue
```

The type identifies the handler class; the other properties depend on the type of handler created. If a separate properties file is used to initialize the handler (such as the JMS handlers), the properties file is found in the classpath. For example, if properties file is at: `{gg_install_dir}/dirprm/foo.properties`, then specify in the properties file as follows: `gg.handler.name.properties=foo.properties`.

10.2.39.3.2 JMS Handler

The main Java property file identifies active handlers. The JMS handler may optionally use a separate property file for JMS-specific configuration. This allows more than one JMS handler to be configured to run at the same time.

There are examples included for several JMS providers (JBoss, TIBCO, Solace, ActiveMQ, WebLogic). For a specific JMS provider, you can choose the appropriate properties files as a starting point for your environment. Each JMS provider has slightly different settings, and your environment will have unique settings as well.

The installation directory for the Java JARs (ggjava) contains the core application JARs (ggjava.jar) and its dependencies in `resources/lib/*.jar`. The resources directory contains all dependencies and configuration, and is in the classpath.

If the JMS client JARs already exist somewhere on the system, they can be referenced directly and added to the classpath without copying them.

The following types of JMS handlers can be specified:

- **jms** – sends text messages to a topic or queue. The messages may be formatted using Velocity templates or by writing a formatter in Java. The same formatters can be used for a `jms_text` message as for writing to files. (`jms_text` is a synonym for `jms`.)

- **aq** – sends text messages to Oracle Advanced Queuing (AQ). The aq handler is a jms handler configured for delivery to AQ. The messages can be formatted using Velocity templates or a custom formatter.
- **jms_map** – sends a JMS MapMessage to a topic or queue. The JMSType of the message is set to the name of the table. The body of the message consists of the following metadata, followed by column name and column value pairs:
 - GG_ID – position of the record, uniquely identifies this operation
 - GG_OPTYPE – type of SQL (insert/update/delete),
 - GG_TABLE – table name on which the operation occurred
 - GG_TX_TIMESTAMP – timestamp of the operation

10.2.39.3.3 File Handler

The file handler is often used to verify the message format when the actual target is JMS, and the message format is being developed using custom Java or Velocity templates. Here is a property file using a file handler:

```
# one file handler active, using Velocity template formatting
gg.handlerlist=myfile
gg.handler.myfile.type=file
gg.handler.myfile.rollover.size=5M
gg.handler.myfile.format=sample2xml.vm
gg.handler.myfile.file=output.xml
```

This example uses a single handler (though, a JMS handler and the file handler could be used at the same time), writing to a file called `output.xml`, using a Velocity template called `sample2xml.vm`. The template is found using the classpath.

10.2.39.3.4 Custom Handlers

For information on coding a custom handler, see [Coding a Custom Handler in Java](#).

10.2.39.3.5 Formatting the Output

As previously described, the existing JMS and file output handlers can be configured through the properties file. Each handler has its own specific properties that can be set: for example, the output file can be set for the file handler, and the JMS destination can be set for the JMS handler. Both of these handlers may also specify a custom formatter. The same formatter may be used for both handlers. As an alternative to writing Java code for custom formatting, a Velocity template may be specified. For further information, see [Filtering Events](#).

10.2.39.3.6 Reporting

Summary statistics about the throughput and amount of data processed are generated when the Replicat or Extract process stops. Additionally, statistics can be written periodically either after a specified amount of time or after a specified number of records have been processed. If both time and number of records are specified, then the report is generated for whichever event happens first. These statistical summaries are written to the Oracle GoldenGate report file and the log files.

10.2.39.4 Java Delivery Properties

- [Common Properties](#)

- [Delivery Properties](#)
- [Java Application Properties](#)

10.2.39.4.1 Common Properties

The following properties are common to Java Delivery using either Replicat or Extract.

- [Logging Properties](#)
- [JVM Boot Options](#)

10.2.39.4.1.1 Logging Properties

Logging is controlled by the following properties.

- [gg.log](#)
- [gg.log.level](#)
- [gg.log.file](#)
- [gg.log.classpath](#)

10.2.39.4.1.1.1 gg.log

Specifies the type of logging that is to be used. The default implementation for the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) is the `jdk` option. This is the built-in Java logging called `java.util.logging` (JUL). The other logging options are `log4j` or `logback`.

For example, to set the type of logging to `log4j`:

```
gg.log=log4j
```

The recommended setting is `log4j`. The log file is created in the `dirrpt` subdirectory of the installation. The default log file name includes the group name of the associated Extract and the file extension is `.log`.

```
<process name>_<log level>_log4j.log
```

Therefore if the Oracle GoldenGate Replicat process is called `javaue`, and the `gg.log.level` is set to `debug`, the resulting log file name will be:

```
javaue_debug_log4j.log
```

10.2.39.4.1.1.2 gg.log.level

Specifies the overall log level for all modules. The syntax is:

```
gg.log.level={ERROR|WARN|INFO|DEBUG|TRACE}
```

The log levels are defined as follows:

- **ERROR** – Only write messages if errors occur
- **WARN** – Write error and warning messages
- **INFO** – Write error, warning and informational messages
- **DEBUG** – Write all messages, including debug ones.
- **TRACE** - Highest level of logging, includes all messages.

The default logging level is `INFO`. The messages in this case will be produced on startup, shutdown and periodically during operation. If the level is switched to `DEBUG`, large volumes of

messages may occur which could impact performance. For example, the following sets the global logging level to INFO:

```
# global logging level
gg.log.level=INFO
```

10.2.39.4.1.1.3 gg.log.file

Specifies the path to the log file. The syntax is:

```
gg.log.file=path_to_file
```

Where the *path_to_file* is the fully defined location of the log file. This allows a change to the name of the log, but you must include the Replicat name if you have more than one Replicat to avoid one overwriting the log of the other.

10.2.39.4.1.1.4 gg.log.classpath

Specifies the classpath to the JARs used to implement logging. This configuration property is not typically used as the ggjava.jar library includes the required logging dependency libraries.

```
gg.log.classpath=path_to_jars
```

10.2.39.4.1.2 JVM Boot Options

The following options configure the Java Runtime Environment. Java classpath and memory options are configurable.

- [jvm.bootoptions](#)

10.2.39.4.1.2.1 jvm.bootoptions

Specifies the initial Java classpath and other boot options that will be applied when the JVM starts. The java.class.path needs colon (:) separators for UNIX/Linux and semicolons (;) for Windows. This is where to specify various options for the JVM, such as initial and maximum heap size and classpath; for example:

- **-Xms**: initial java heap size
- **-Xmx**: maximum java heap size
- **-Djava.class.path**: classpath specifying location of at least the main application JAR, ggjava.jar. Other JARs, such as JMS provider JARs, may also be specified here as well; alternatively, these may be specified in the Java application properties file. If using a separate log4j properties file then the location of the properties file must be included in the bootoptions java.class.path included in the bootoptions variable.
- **-verbose:jni**: run in verbose mode (for JNI)

For example (all on a single line):

```
jvm.bootoptions= -Djava.class.path=ggjava/ggjava.jar
-Dlog4j.configuration=my-log4j.properties -Xmx512m
```

The log4j.configuration property identifies a log4j properties file that is resolved by searching the classpath. You may use your own log4j configuration, or one of the preconfigured log4j settings: log4j.properties (default level of logging), debug-log4j.properties (debug logging) or trace-log4j.properties (very verbose logging). To use log4j logging with the Replicat process gg.log=log4j must be set.

Use of the one of the preconfigured log4j settings does not require any change to the classpath since those files are already included in the classpath. The -Djava.class.path variable must include the path to the directory containing a custom log4j configuration file without the * wild card appended.

10.2.39.4.2 Delivery Properties

The following properties are available to Java Delivery:

- [General Properties](#)
- [Statistics and Reporting](#)
Disables or enables the checkpoint file handling. This causes the standard Oracle GoldenGate reporting to be incomplete. Oracle GoldenGate for Java adds its own reporting to handle this issue.

10.2.39.4.2.1 General Properties

The following properties apply to all writer configurations:

- [goldengate.userexit.writers](#)
- [goldengate.userexit.chkptprefix](#)
- [goldengate.userexit.nochkpt](#)
- [goldengate.userexit.usetargetcols](#)

10.2.39.4.2.1.1 goldengate.userexit.writers

Specifies the name of the writer. This is always `jvm` and should not be modified.

For example:

```
goldengate.userexit.writers=jvm
```

All other properties in the file should be prefixed by the writer name, `jvm`.

10.2.39.4.2.1.2 goldengate.userexit.chkptprefix

Specifies a string value for the prefix added to the Java checkpoint file name. For example:

```
goldengate.userexit.chkptprefix=javaue_
```

10.2.39.4.2.1.3 goldengate.userexit.nochkpt

Disables or enables the checkpoint file. The default is `false`, the checkpoint file is enabled. Set this property to `true` if transactions are supported and enabled on the target.

For example, Java Adapter Properties if JMS is the target and JMS local transactions are enabled (the default), set `goldengate.userexit.nochkpt=true` to disable the user exit checkpoint file. If JMS transactions are disabled by setting `localTx=false` on the handler, the checkpoint file should be enabled by setting `goldengate.userexit.nochkpt=false`.

```
goldengate.userexit.nochkpt=true|false
```

10.2.39.4.2.1.4 goldengate.userexit.usetargetcols

Specifies whether or not mapping to target columns is allowed. The default is `false`, no target mapping.

```
goldengate.userexit.usetargetcols=true|false
```

10.2.39.4.2.2 Statistics and Reporting

Disables or enables the checkpoint file handling. This causes the standard Oracle GoldenGate reporting to be incomplete. Oracle GoldenGate for Java adds its own reporting to handle this issue.

Statistics can be reported every `t` seconds or every `n` records - or if both are specified, whichever criteria is met first.

There are two sets of statistics recorded: those maintained by the Replicat module and those obtained from the Java module. The reports received from the Java side are formatted and returned by the individual handlers.

The statistics include the total number of operations, transactions and corresponding rates.

- [jvm.stats.display](#)
- [jvm.stats.full](#)
- [jvm.stats.time | jvm.stats.numrecs](#)

10.2.39.4.2.2.1 *jvm.stats.display*

Controls the output of statistics to the Oracle GoldenGate report file and to the user exit log files.

The following example outputs these statistics.

```
jvm.stats.display=true
```

10.2.39.4.2.2.2 *jvm.stats.full*

Controls the output of statistics from the Java side, in addition to the statistics from the C side.

Java side statistics are more detailed but also involve some additional overhead, so if statistics are reported often and a less detailed summary is adequate, it is recommended that `stats.full` property is set to `false`.

The following example will output Java statistics in addition to C.

```
jvm.stats.full=true
```

10.2.39.4.2.2.3 *jvm.stats.time | jvm.stats.numrecs*

Specifies a time interval, in seconds or a number of records, after which statistics will be reported. The default is to report statistics every hour or every 10000 records (which ever occurs first).

For example, to report every 10 minutes or every 1000 records, specify:

```
jvm.stats.time=600  
jvm.stats.numrecs=1000
```

The Java application statistics are handler-dependent:

- For the all handlers, there is at least the total elapsed time, processing time, number of operations, transactions;
- For the JMS handler, there is additionally the total number of bytes received and sent.
- The report can be customized using a template.

10.2.39.4.3 Java Application Properties

The following defines the properties which may be set in the Java application property file.

- [Properties for All Handlers](#)
- [Properties for Formatted Output](#)
- [Properties for CSV and Fixed Format Output](#)
- [File Writer Properties](#)
- [JMS Handler Properties](#)
- [JNDI Properties](#)

- [General Properties](#)
- [Java Delivery Transaction Grouping](#)

10.2.39.4.3.1 Properties for All Handlers

The following properties apply to all handlers:

- [gg.handlerlist](#)
- [gg.handler.name.type](#)

10.2.39.4.3.1.1 gg.handlerlist

The handler list is a list of active handlers separated by commas. These values are used in the rest of the property file to configure the individual handlers. For example:

```
gg.handlerlist=name1, name2
gg.handler.name1.propertyA=value1
gg.handler.name1.propertyB=value2
gg.handler.name1.propertyC=value3
gg.handler.name2.propertyA=value1
gg.handler.name2.propertyB=value2
gg.handler.name2.propertyC=value3
```

Using the `handlerlist` property, you may include completely configured handlers in the property file and just disable them by removing them from the handler list.

10.2.39.4.3.1.2 gg.handler.name.type

This type of handler. This is either a predefined value for built-in handlers, or a fully qualified Java class name. The syntax is:

```
gg.handler.name.type={jms|jms_map|aq|singlefile|rollingfile|custom_java_class}
```

Where:

All but the last are pre-defined handlers:

- **jms** – Sends transactions, operations, and metadata as formatted messages to a JMS provider
- **aq** – Sends transactions, operations, and metadata as formatted messages to Oracle Advanced Queuing (AQ)
- **jms_map** – Sends JMS map messages
- **singlefile** – Writes to a single file on disk, but does not roll the file
- **rollingfile** – Writes transactions, operations, and metadata to a file on disk, rolling the file over after a certain size, amount of time, or both. For example:

```
gg.handler.name1.rolloverSize=5000000
gg.handler.name1.rolloverTime=1m
```

- **custom_java_class** – Any class that extends the Oracle GoldenGate for Java `AbstractHandler` class and can handle transaction, operation, or metadata events

The Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) package also contains more predefined handlers to write to various GG for DAA targets.

10.2.39.4.3.2 Properties for Formatted Output

The following properties apply to all handlers capable of producing formatted output; this includes:

- The `jms_text` handler (but not the `jms_map` handler)
- The `aq` handler
- The `singlefile` and `rolling` handlers, for writing formatted output to files
- The predefined Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) handlers
- [gg.handler.name.format](#)
- [gg.handler.name.includeTables](#)
- [gg.handler.name.excludeTables](#)
- [gg.handler.name.mode](#), [gg.handler.name.format.mode](#)

10.2.39.4.3.2.1 `gg.handler.name.format`

Specifies the format used to transform operations and transactions into messages sent to JMS, to the Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) target or to a file. The format is specified uniquely for each handler. The value may be:

- **Velocity template**
- **Java class name** (fully qualified - the class specified must be a type of formatter)
- **csv** for delimited values (such as comma separated values; the delimiter can be customized)
- **fixed** for fixed-length fields
- **Built-in formatter**, such as:
 - `xml` – demo XML format
 - `xml2` – internal XML format

For example, to specify a custom Java class:

```
gg.handlerlist=abc
gg.handler.abc.format=com.mycompany.MyFormat
```

Or, for a Velocity template:

```
gg.handlerlist=xyz
gg.handler.xyz.format=path/to/sample.vm
```

If using templates, the file is found relative to some directory or JAR that is in the classpath. By default, the Oracle GoldenGate installation directory is in the classpath, so the preceding template could be placed in the `dirprm` directory of the Oracle GoldenGate installation location.

The default format is to use the built-in XML formatter.

10.2.39.4.3.2.2 `gg.handler.name.includeTables`

Specifies a list of tables this handler will include.

If the schema (or owner) of the table is specified, then only that schema matches the table name; otherwise, the table name matches any schema. A comma separated list of tables can be specified. For example, to have the handler only process tables `foo.customer` and `bar.orders`:

```
gg.handler.myhandler.includeTables=foo.customer, bar.orders
```

If the catalog and schema (or owner) of the table are specified, then only that catalog and schema matches the table name; otherwise, the table name matches any catalog and schema.

A comma separated list of tables can be specified. For example, to have the handler only process tables `dbo.foo.customer` and `dbo.bar.orders`:

```
gg.handler.myhandler.includeTables=dbo.foo.customer, dbo.bar.orders
```

Note

In order to selectively process operations on a table by table basis, the handler must be processing in operation mode. If the handler is processing in transaction mode, then when a single transaction contains several operations spanning several tables, if any table matches the include list of tables, the transaction will be included.

10.2.39.4.3.2.3 `gg.handler.name.excludeTables`

Specifies a list of tables this handler will exclude.

If the schema (or owner) of the table is specified, then only that schema matches the table name; otherwise, the table name matches any schema. A list of tables may be specified, comma-separated. For example, to have the handler process all operations on all tables except table `date_modified` in all schemas:

```
gg.handler.myhandler.excludeTables=date_modified
```

If the catalog and schema (or owner) of the table are specified, then only that catalog and schema matches the table name; otherwise, the table name matches any catalog and schema. A list of tables may be specified, comma-separated. For example, to have the handler process all operations on all tables except table `date_modified` in catalog `dbo` and schema `bar`:

```
gg.handler.myhandler.excludeTables=dbo.bar.date_modified
```

10.2.39.4.3.2.4 `gg.handler.name.mode`, `gg.handler.name.format.mode`

Specifies whether to output one operation per message (op) or one transaction per message (tx). The default is op. Use `gg.handler.name.format.mode` when you have a custom formatter.

10.2.39.4.3.3 Properties for CSV and Fixed Format Output

If the handler is set to use either comma separated values (CSV) or fixed format output, the following properties may also be set.

- [`gg.handler.name.format.delim`](#)
- [`gg.handler.name.format.quote`](#)
- [`gg.handler.name.format.metacols`](#)
- [`gg.handler.name.format.missingColumnChar`](#)
- [`gg.handler.name.format.presentColumnChar`](#)
- [`gg.handler.name.format.nullColumnChar`](#)
- [`gg.handler.name.format.beginTxChar`](#)
- [`gg.handler.name.format.middleTxChar`](#)
- [`gg.handler.name.format.endTxChar`](#)
- [`gg.handler.name.format.wholeTxChar`](#)
- [`gg.handler.name.format.insertChar`](#)
- [`gg.handler.name.format.updateChar`](#)

- [gg.handler.name.format.deleteChar](#)
- [gg.handler.name.format.truncateChar](#)
- [gg.handler.name.format.endOfLine](#)
- [gg.handler.name.format.justify](#)
- [gg.handler.name.format.includeBefore](#)

10.2.39.4.3.3.1 gg.handler.name.format.delim

Specifies the delimiter to use between fields. Set this to no value to have no delimiter used. For example:

```
gg.handler.handler1.format.delim=,
```

10.2.39.4.3.3.2 gg.handler.name.format.quote

Specifies the quote character to be used if column values are quoted. For example:

```
gg.handler.handler1.format.quote='
```

10.2.39.4.3.3.3 gg.handler.name.format.metacols

Specifies the metadata column values to appear at the beginning of the record, before any column data. Specify any of the following, in the order they should appear:

- **position** – unique position indicator of records in a trail
- **opcode** – I, U, or D for insert, update, or delete records (see: `insertChar`, `updateChar`, `deleteChar`)
- **txind** – transaction indicator – such as 0=begin, 1=middle, 2=end, 3=whole tx (see `beginTxChar`, `middleTxChar`, `endTxChar`, `wholeTxChar`)
- **opcount** – position of a record in a transaction, starting from 0
- **catalog** – catalog of the schema for the record
- **schema** – schema/owner of the table for the record
- **tableonly** – just table (no schema/owner)
- **table** – full name of table, `catalog.schema.table`
- **timestamp** – commit timestamp of record

For example:

```
gg.handler.handler1.format.metacols=opcode, table, txind, position
```

10.2.39.4.3.3.4 gg.handler.name.format.missingColumnChar

Specifies a special column prefix for a column value that was not captured from the source database transaction log. The column value is not in trail and it is unknown if it has a value or is NULL

The character used to represent the missing state of the column value can be customized. For example:

```
gg.handler.handler1.format.missingColumnChar=M
```

By default, the missing column value is set to an empty string and does not show.

10.2.39.4.3.3.5 gg.handler.name.format.presentColumnChar

Specifies a special column prefix for a column value that exists in the trail and is not NULL.

The character used to represent the state of the column can be customized. For example:

```
gg.handler.handler1.format.presentColumnChar=P
```

By default, the present column value is set to an empty string and does not show.

10.2.39.4.3.3.6 `gg.handler.name.format.nullColumnChar`

Specifies a special column prefix for a column value that exists in the trail and is set to NULL.

The character used to represent the state of the column can be customized. For example:

```
gg.handler.handler1.format.nullColumnChar=N
```

By default, the null column value is set to an empty string and does not show.

10.2.39.4.3.3.7 `gg.handler.name.format.beginTxChar`

Specifies the header metadata character (see `metacols`) used to identify a record as the begin of a transaction. For example:

```
gg.handler.handler1.format.beginTxChar=B
```

10.2.39.4.3.3.8 `gg.handler.name.format.middleTxChar`

Specifies the header metadata characters (see `metacols`) used to identify a record as the middle of a transaction. For example:

```
gg.handler.handler1.format.middleTxChar=M
```

10.2.39.4.3.3.9 `gg.handler.name.format.endTxChar`

Specifies the header metadata characters (see `metacols`) used to identify a record as the end of a transaction. For example:

```
gg.handler.handler1.format.endTxChar=E
```

10.2.39.4.3.3.10 `gg.handler.name.format.wholeTxChar`

Specifies the header metadata characters (see `metacols`) used to identify a record as a complete transaction; referred to as a whole transaction. For example:

```
gg.handler.handler1.format.wholeTxChar=W
```

10.2.39.4.3.3.11 `gg.handler.name.format.insertChar`

Specifies the character to identify an insert operation. The default is I.

For example, to use INS instead of I for insert operations:

```
gg.handler.handler1.format.insertChar=INS
```

10.2.39.4.3.3.12 `gg.handler.name.format.updateChar`

Specifies the character to identify an update operation. The default is U.

For example, to use UPD instead of U for update operations:

```
gg.handler.handler1.format.updateChar=UPD
```

10.2.39.4.3.3.13 `gg.handler.name.format.deleteChar`

Specifies the character to identify a delete operation. The default is D.

For example, to use DEL instead of D for delete operations:

```
gg.handler.handler1.format.deleteChar=DEL
```

10.2.39.4.3.3.14 `gg.handler.name.format.truncateChar`

Specifies the character to identify a truncate operation. The default is T.

For example, to use TRUNC instead of T for truncate operations:

```
gg.handler.handler1.format.truncateChar=TRUNC
```

10.2.39.4.3.3.15 `gg.handler.name.format.endOfLine`
Specifies the end-of-line character as:

- EOL - Native platform
- CR - Neutral (UNIX-style \n)
- CRLF - Windows (\r\n)

For example:

```
gg.handler.handler1.format.endOfLine=CR
```

10.2.39.4.3.3.16 `gg.handler.name.format.justify`
Specifies the left or right justification of fixed fields. For example:

```
gg.handler.handler1.format.justify=left
```

10.2.39.4.3.3.17 `gg.handler.name.format.includeBefores`
Controls whether before images should be included in the output. There must be before images in the trail. For example:

```
gg.handler.handler1.format.includeBefores=false
```

10.2.39.4.3.4 File Writer Properties

The following properties only apply to handlers that write their output to files: the file handler and the singlefile handler.

- [gg.handler.name.file](#)
- [gg.handler.name.append](#)
- [gg.handler.name.rolloverSize](#)

10.2.39.4.3.4.1 `gg.handler.name.file`
Specifies the name of the output file for the given handler. If the handler is a rolling file, this name is used to derive the rolled file names. The default file name is `output.xml`.

10.2.39.4.3.4.2 `gg.handler.name.append`
Controls whether the file should be appended to (`true`) or overwritten upon restart (`false`).

10.2.39.4.3.4.3 `gg.handler.name.rolloverSize`
If using the file handler, this specifies the size of the file before a rollover should be attempted. The file size will be at least this size, but will most likely be larger. Operations and transactions are not broken across files. The size is specified in bytes, but a suffix may be given to identify MB or KB. For example:

```
gg.handler.myfile.rolloverSize=5MB
```

The default rollover size is 10MB.

10.2.39.4.3.5 JMS Handler Properties

The following properties apply to the JMS handlers. Some of these values may be defined in the Java application properties file using the name of the handler. Other properties may be placed into a separate JMS properties file, which is useful if using more than one JMS handler at a time. For example:

```
gg.handler.myjms.type=jms_text
gg.handler.myjms.format=xml
gg.handler.myjms.properties=weblogic.properties
```

Just as with Velocity templates and formatting property files, this additional JMS properties file is found in the classpath. The preceding properties file `weblogic.properties` would be found in `{gg_install_dir}/dirprm/weblogic.properties`, since the `dirprm` directory is included by default in the classpath.

Settings that can be made in the Java application properties file will override the corresponding value set in the supplemental JMS properties file (`weblogic.properties` in the preceding example). In the following example, the destination property is specified in the Java application properties file. This allows the same default connection information for the two handlers `myjms1` and `myjms2`, but customizes the target destination queue.

```
gg.handlerlist=myjms1,myjms2
gg.handler.myjms1.type=jms_text
gg.handler.myjms1.destination=queue.sampleA
gg.handler.myjms1.format=sample.vm
gg.handler.myjms1.properties=tibco-default.properties
gg.handler.myjms2.type=jms_map
gg.handler.myjms2.destination=queue.sampleB
gg.handler.myjms2.properties=tibco-default.properties
```

To set a property, specify the handler name as a prefix; for example:

```
gg.handlerlist=sample
gg.handler.sample.type=jms_text
gg.handler.sample.format=my_template.vm
gg.handler.sample.destination=gg.myqueue
gg.handler.sample.queueortopic=queue
gg.handler.sample.connectionUrl=tcp://host:61616?jms.useAsyncSend=true
gg.handler.sample.useJndi=false
gg.handler.sample.connectionFactory=ConnectionFactory
gg.handler.sample.connectionFactoryClass=\
    org.apache.activemq.ActiveMQConnectionFactory
gg.handler.sample.timeToLive=50000
```

- [Standard JMS Settings](#)
- [Group Transaction Properties](#)

10.2.39.4.3.5.1 Standard JMS Settings

The following outlines the JMS properties which may be set, and the accepted values. These apply for both JMS handler types: `jms_text` (TextMessage) and `jms_map` (MapMessage).

- [gg.handler.name.destination](#)
- [gg.handler.name.user](#)
- [gg.handler.name.password](#)
- [gg.handler.name.queueOrTopic](#)
- [gg.handler.name.persistent](#)
- [gg.handler.name.priority](#)
- [gg.handler.name.timeToLive](#)
- [gg.handler.name.connectionFactory](#)
- [gg.handler.name.useJndi](#)
- [gg.handler.name.connectionUrl](#)

- [`gg.handler.name.connectionFactoryClass`](#)
- [`gg.handler.name.localTX`](#)
- [`gg.handlerlist.nop`](#)
- [`gg.handler.name.physicalDestination`](#)

10.2.39.4.3.5.1.1 `gg.handler.name.destination`

The queue or topic to which the message is sent. This must be correctly configured on the JMS server. Typical values may be: `queue/A`, `queue.Test`, `example.MyTopic`, etc.

```
gg.handler.name.destination=queue_or_topic
```

10.2.39.4.3.5.1.2 `gg.handler.name.user`

(Optional) User name required to send messages to the JMS server.

```
gg.handler.name.user=user_name
```

10.2.39.4.3.5.1.3 `gg.handler.name.password`

(Optional) Password required to send messages to the JMS server

```
gg.handler.name.password=password
```

10.2.39.4.3.5.1.4 `gg.handler.name.queueOrTopic`

Whether the handler is sending to a queue (a single receiver) or a topic (publish / subscribe). This must be correctly configured in the JMS provider. This property is an alias of `gg.handler.name.destination`. The syntax is:

```
gg.handler.name.queueOrTopic=queue|topic
```

Where:

- `queue` – a message is removed from the queue once it has been read. This is the default.
- `topic` – messages are published and may be delivered to multiple subscribers.

10.2.39.4.3.5.1.5 `gg.handler.name.persistent`

If the delivery mode is set to persistent or not. If the messages are to be persistent, the JMS provider must be configured to log the message to stable storage as part of the client's send operation. The syntax is:

```
gg.handler.name.persistent={true|false}
```

10.2.39.4.3.5.1.6 `gg.handler.name.priority`

JMS defines a 10 level priority value, with 0 as the lowest and 9 as the highest. Priority is set to 4 by default. The syntax is:

```
gg.handler.name.priority=integer
```

For example:

```
gg.handler.name.priority=5
```

10.2.39.4.3.5.1.7 `gg.handler.name.timeToLive`

The length of time in milliseconds from its dispatch time that a produced message should be retained by the message system. A value of zero specifies the time is unlimited. The default is zero. The syntax is:

```
gg.handler.name.timeToLive=milliseconds
```

For example:

```
gg.handler.name.timeToLive= 36000
```

10.2.39.4.3.5.1.8 gg.handler.name.connectionFactory

Name of the connection factory to lookup using JNDI. ConnectionFactoryJNDIName is an alias. The syntax is:

```
gg.handler.name.connectionFactory=JNDI_name
```

10.2.39.4.3.5.1.9 gg.handler.name.useJndi

If gg.handler.name.usejndi is false, then JNDI is not used to configure the JMS client. Instead, factories and connections are explicitly constructed. The syntax is:

```
gg.handler.name.useJndi=true|false
```

10.2.39.4.3.5.1.10 gg.handler.name.connectionUrl

Connection URL is used only when not using JNDI to explicitly create the connection. The syntax is:

```
gg.handler.name.connectionUrl=url
```

10.2.39.4.3.5.1.11 gg.handler.name.connectionFactoryClass

The Connection Factory Class is used to access a factory only when not using JNDI. The value of this property is the Java class name to instantiate; constructing a factory object explicitly.

```
gg.handler.name.connectionFactoryClass=java_class_name
```

10.2.39.4.3.5.1.12 gg.handler.name.localTX

Specifies whether or not local transactions are used. The default is true, local transactions are used. The syntax is:

```
gg.handler.name.localTX=true|false
```

10.2.39.4.3.5.1.13 gg.handlerlist.nop

Disables the sending of JMS messages to allow testing of message generation. This is a global property used only for testing. The events are still generated and handled and the message is constructed. The default is false; do not disable message send. The syntax is:

```
gg.handlerlist.nop=true|false
```

Users can take advantage of this option to measure the performance of trail records processing without involving the handler module. This approach can narrow down the possible culprits of a suspected performance issue while applying trail records to the target system.

10.2.39.4.3.5.1.14 gg.handler.name.physicalDestination

Name of the queue or topic object, obtained through the ConnectionFactory API instead of the JNDI provider.

```
gg.handler.name.physicalDestination=queue_name
```

10.2.39.4.3.5.2 Group Transaction Properties

These properties set limits for grouping transactions.

10.2.39.4.3.6 JNDI Properties

These JNDI properties are required for connection to an Initial Context to look up the connection factory and initial destination.

```
java.naming.provider.url=url
java.naming.factory.initial=java-class-name
```

If JNDI security is enabled, the following properties may be set:

```
java.naming.security.principal=user-name  
java.naming.security.credentials=password-or-other-authenticator
```

For example:

```
java.naming.provider.url= t3://localhost:7001  
java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory  
java.naming.security.principal=jndiuser  
java.naming.security.credentials=jndipw
```

10.2.39.4.3.7 General Properties

The following are general properties that are used for the Java framework:

- [gg.classpath](#)
- [gg.report.time](#)
- [gg.binaryencoding](#)

10.2.39.4.3.7.1 gg.classpath

Specifies a comma delimited list of additional paths to directories or JARs to add to the classpath. Optionally, the list can be delimited by semicolons for Windows systems or by colons for UNIX. For example:

```
gg.classpath=C:\Program Files\MyProgram\bin;C:\Program Files\ProgramB\app\bin;
```

This Adapter properties file configuration property should be used to configure pathing to custom Java JARs or to the external dependencies of Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA).

10.2.39.4.3.7.2 gg.report.time

Specifies how often statistics are calculated and sent to Extract for the processing report. If Extract is configured to print a report, these statistics are included. The syntax is:

```
gg.report.time=report_interval{s|m|h}
```

Where:

- *report_interval* is an integer
- The valid time units are:
 - s - seconds
 - m - minutes
 - h - hours

If no value is entered, the default is to calculate and send every 24 hours.

10.2.39.4.3.7.3 gg.binaryencoding

Specifies the binary encoding type. The desired output encoding for binary data can be configured using this property. For example:

```
gg.binaryencoding=base64|hex
```

The default value is base64. The valid values to represent binary data are:

- base64 - a base64 string
- hex - a hexadecimal string

10.2.39.4.3.8 Java Delivery Transaction Grouping

Transaction grouping can significantly improve the performance of Java integrations especially Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) integrations. Java Delivery provides functionality to perform transaction grouping. When Java Delivery is hosted by a Replicat process then the GROUPTRANSOPS Replicat configuration should be used to perform transaction grouping.

10.2.39.5 Developing Custom Filters, Formatters, and Handlers

- [Filtering Events](#)
- [Custom Formatting](#)
- [Coding a Custom Handler in Java](#)
- [Additional Resources](#)

10.2.39.5.1 Filtering Events

By default, all transactions, operations and metadata events are passed to the DataSourceListener event handlers. An event filter can be implemented to filter the events sent to the handlers. The filter could select certain operations on certain tables containing certain column values, for example

Filters are additive: if more than one filter is set for a handler, then all filters must return true in order for the event to be passed to the handler.

You can configure filters using the Java application properties file:

```
# handler "foo" only receives certain events
gg.handler.one.type=jms
gg.handler.one.format=mytemplate.vm
gg.handler.one.filter=com.mycompany.MyFilter
```

To activate the filter, you write the filter and set it on the handler; no additional logic needs to be added to specific handlers.

10.2.39.5.2 Custom Formatting

You can customize the output format of a built-in handler by:

- Writing a custom formatter in Java or
- Using a velocity template
- [Coding a Custom Formatter in Java](#)
- [Using a Velocity Template](#)

10.2.39.5.2.1 Coding a Custom Formatter in Java

The preceding examples show a JMS handler and a file output handler using the same formatter (com.mycompany.MyFormatter). The following is an example of how this formatter may be implemented.

Example 10-2 Custom Formatting Implementation

```
package com.mycompany.MyFormatter;
import oracle.goldengate.datasource.DsOperation;
import oracle.goldengate.datasource.DsTransaction;
```

```

import oracle.goldengate.datasource.format.DsFormatterAdapter;
import oracle.goldengate.datasource.meta.ColumnMetaData;
import oracle.goldengate.datasource.meta.DsMetaData;
import oracle.goldengate.datasource.meta.TableMetaData;
import java.io.PrintWriter;
public class MyFormatter extends DsFormatterAdapter {

    public MyFormatter() { }
    @Override
    public void formatTx(DsTransaction tx,

DsMetaData meta,
PrintWriter out)

    {

        out.print("Transaction: " );
        out.print("numOps=\'" + tx.getSize() + "\' " );
        out.println("ts=\'" + tx.getStartTxTimeAsString() + "\'");
        for(DsOperation op: tx.getOperations()) {
            TableName currTable = op.getTableName();
            TableMetaData tMeta = dbMeta.getTableMetaData(currTable);
            String opType = op.getOperationType().toString();
            String table = tMeta.getTableName().getFullName();
            out.println(opType + " on table \'\" + table + "\"");
            int colNum = 0;
            for(DsColumn col: op.getColumns())
            {

                ColumnMetaData cMeta = tMeta.getColumnMetaData( colNum++ );
                out.println(
                    cMeta.getColumnName() + " = " + col.getAfterValue() );
            }

        }
    }
    @Override
    public void formatOp(DsTransaction tx,

DsOperation op,
TableMetaData tMeta,
PrintWriter out)

    {

        // not used...

    }

}

```

The formatter defines methods for either formatting complete transactions (after they are committed) or individual operations (as they are received, before the commit). If the formatter is in operation mode, then `formatOp(...)` is called; otherwise, `formatTx(...)` is called at transaction commit.

To compile and use this custom formatter, include the Oracle GoldenGate for Java JARs in the classpath and place the compiled `.class` files in `gg_install_dir/dirprm`:

```

javac -d gg_install_dir/dirprm
-classpath ggjava/ggjava.jar MyFormatter.java

```

The resulting class files are located in `resources/classes` (in correct package structure):

```
gg_install_dir/dirprm/com/mycompany/MyFormatter.class
```

Alternatively, the custom classes can be put into a JAR; in this case, either include the JAR file in the JVM classpath using the user exit properties (using `java.class.path` in the `jvm.bootoptions` property), or by setting the Java application properties file to include your custom JAR:

```
# set properties on 'one'
gg.handler.one.type=file
gg.handler.one.format=com.mycompany.MyFormatter
gg.handler.one.file=output.xml
gg.classpath=/path/to/my.jar,/path/to/directory/of/jars/*
```

10.2.39.5.2.2 Using a Velocity Template

As an alternative to writing Java code for custom formatting, Velocity templates can be a good alternative to quickly prototype formatters. For example, the following template could be specified as the format of a JMS or file handler:

```
Transaction: numOps='$tx.size' ts='$tx.timestamp'
#for each( $op in $tx )
operation: $op.sqlType, on table "$op.tableName":
#for each( $col in $op )
$op.tableName, $col.meta.columnName = $col.value
#end
#end
```

If the template were named `sample.vm`, it could be placed in the classpath, for example:

```
gg_install_dir/dirprm/sample.vm
```

Update the Java application properties file to use the template:

```
# set properties on 'one'
gg.handler.one.type=file
gg.handler.one.format=sample.vm
gg.handler.one.file=output.xml
```

When modifying templates, there is no need to recompile any Java source; simply save the template and re-run the Java application. When the application is run, the following output would be generated (assuming a table named `SCHEMA.SOMETABLE`, with columns `TESTCOLA` and `TESTCOLB`):

```
Transaction: numOps='3' ts='2008-12-31 12:34:56.000'
operation: UPDATE, on table "SCHEMA.SOMETABLE":
SCHEMA.SOMETABLE, TESTCOLA = value 123
SCHEMA.SOMETABLE, TESTCOLB = value abc
operation: UPDATE, on table "SCHEMA.SOMETABLE":
SCHEMA.SOMETABLE, TESTCOLA = value 456
SCHEMA.SOMETABLE, TESTCOLB = value def
operation: UPDATE, on table "SCHEMA.SOMETABLE":
SCHEMA.SOMETABLE, TESTCOLA = value 789
SCHEMA.SOMETABLE, TESTCOLB = value ghi
```

10.2.39.5.3 Coding a Custom Handler in Java

A custom handler can be implemented by extending `AbstractHandler` as in the following example:

```

import oracle.goldengate.datasource.*;
import static oracle.goldengate.datasource.GGDataSource.Status;
public class SampleHandler extends AbstractHandler {
    @Override
    public void init(DsConfiguration conf, DsMetaData metaData) {
        super.init(conf, metaData);
        // ... do additional config...
    }
    @Override
    public Status operationAdded(DsEvent e, DsTransaction tx, DsOperation op) { ... }
    @Override
    public Status transactionCommit(DsEvent e, DsTransaction tx) { ... }
    @Override
    public Status metaDataChanged(DsEvent e, DsMetaData meta) { .... }
    @Override
    public void destroy() { /* ... do cleanup ... */ }
    @Override
    public String reportStatus() { return "status report..."; }
    @Override
    public Status ddlOperation(OpType opType, ObjectType objectType, String
objectName, String ddlText) }

```

The method in `AbstractHandler` is not abstract rather it has a body. In the body it performs cached metadata invalidation by marking the metadata object as dirty. It also provides TRACE-level logging of DDL events when the `ddlOperation` method is specified. You can override this method in your custom handler implementations. You should always call the super method before any custom handling to ensure the functionality in `AbstractHandler` is executed

When a transaction is processed from the Extract, the order of calls into the handler is as follows:

1. Initialization:
 - First, the handler is constructed.
 - Next, all the "setters" are called on the instance with values from the property file.
 - Finally, the handler is initialized; the `init(...)` method is called before any transactions are received. It is important that the `init(...)` method call `super.init(...)` to properly initialize the base class.
2. Metadata is then received. If the Java module is processing an operation on a table not yet seen during this run, a metadata event is fired, and the `metaDataChanged(...)` method is called. Typically, there is no need to implement this method. The `DsMetaData` is automatically updated with new data source metadata as it is received.
3. A transaction is started. A transaction event is fired, causing the `transactionBegin(...)` method on the handler to be invoked (this is not shown). This is typically not used, since the transaction has zero operations at this point.
4. Operations are added to the transaction, one after another. This causes the `operationAdded(...)` method to be called on the handler for each operation added. The containing transaction is also passed into the method, along with the data source metadata that contains all processed table metadata. The transaction has not yet been committed, and could be aborted before the commit is received.

Each operation contains the column values from the transaction (possibly just the changed values when Extract is processing with compressed updates.) The column values may contain both before and after values.

For the `ddlOperation` method, the options are:

- `opType` - Is an enumeration that identifies the DDL operation type that is occurring (CREATE, ALTER, and so on).
 - `objectType` - Is an enumeration that identifies the type of the target of the DDL (TABLE, VIEW, and so on).
 - `objectName` - Is the fully qualified source object name; typically a fully qualified table name.
 - `ddlText` - Is the raw DDL text executed on the source relational database.
5. The transaction is committed. This causes the `transactionCommit(...)` method to be called.
 6. Periodically, `reportStatus` may be called; it is also called at process shutdown. Typically, this displays the statistics from processing (the number of operations and transactions processed and other details).

An example of a simple printer handler, which just prints out very basic event information for transactions, operations and metadata follows. The handler also has a property `myoutput` for setting the output file name; this can be set in the Java application properties file as follows:

```
gg.handlerlist=sample
# set properties on 'sample'
gg.handler.sample.type=sample.SampleHandler
gg.handler.sample.myoutput=out.txt
```

To use the custom handler,

1. Compile the class
2. Include the class in the application classpath,
3. Add the handler to the list of active handlers in the Java application properties file.

To compile the handler, include the Oracle GoldenGate for Java JARs in the classpath and place the compiled `.class` files in `gg_install_dir/javaue/resources/classes`:

```
javac -d gg_install_dir/dirprm
-classpath ggjava/ggjava.jar SampleHandler.java
```

The resulting class files would be located in `resources/classes`, in correct package structure, such as:

```
gg_install_dir/dirprm/sample/SampleHandler.class
```

Note

For any Java application development beyond *hello world* examples, either Ant or Maven would be used to compile, test and package the application. The examples showing `javac` are for illustration purposes only.

Alternatively, custom classes can be put into a JAR and included in the classpath. Either include the custom JAR files in the JVM classpath using the Java properties (using `java.class.path` in the `jvm.bootoptions` property), or by setting the Java application properties file to include your custom JAR:

```
# set properties on 'one'
gg.handler.one.type=sample.SampleHandler
gg.handler.one.myoutput=out.txt
gg.classpath=/path/to/my.jar,/path/to/directory/of/jars/*
```


The classpath property can be set on any handler to include additional individual JARs, a directory (which would contain resources or extracted class files) or a whole directory of JARs. To include a whole directory of JARs, use the Java 6 style syntax:

```
c:/path/to/directory/* (or on UNIX: /path/to/directory/* )
```

Only the wildcard * can be specified; a file pattern cannot be used. This automatically matches all files in the directory ending with the .jar suffix. To include multiple JARs or multiple directories, you can use the system-specific path separator (on UNIX, the colon and on Windows the semicolon) or you can use platform-independent commas, as shown in the preceding example.

If the handler requires many properties to be set, just include the property in the parameter file, and your handler's corresponding "setter" will be called. For example:

```
gg.handler.one.type=com.mycompany.MyHandler
gg.handler.one.myOutput=out.txt
gg.handler.one.myCustomProperty=12345
```

The preceding example would invoke the following methods in the custom handler:

```
public void setMyOutput(String s) {

    // use the string...

} public void setMyCustomProperty(int j) {

    // use the int...

}
```

Any standard Java type may be used, such as int, long, String, boolean. For custom types, you may create a custom property editor to convert the String to your custom type.

10.2.39.5.4 Additional Resources

There is Javadoc available for the Java API. The Javadoc has been intentionally reduced to a set of core packages, classes and interfaces in order to only distribute the relevant interfaces and classes useful for customizing and extension.

In each package, some classes have been intentionally omitted for clarity. The important classes are:

- `oracle.goldengate.datasource.DsTransaction`: represents a database transaction. A transaction contains zero or more operations.
- `oracle.goldengate.datasource.DsOperation`: represents a database operation (insert, update, delete). An operation contains zero or more column values representing the data-change event. Columns indexes are offset by zero in the Java API.
- `oracle.goldengate.datasource.DsColumn`: represents a column value. A column value is a composite of a before and an after value. A column value may be 'present' (having a value or be null) or 'missing' (is not included in the source trail).
 - `oracle.goldengate.datasource.DsColumnComposite` is the composite
 - `oracle.goldengate.datasource.DsColumnBeforeValue` is the column value before the operation (this is optional, and may not be included in the operation)
 - `oracle.goldengate.datasource.DsColumnAfterValue` is the value after the operation

- `oracle.goldengate.datasource.meta.DsMetaData`: represents all database metadata seen; initially, the object is empty. `DsMetaData` contains a hash map of zero or more instances of `TableMetaData`, using the `TableName` as a key.
- `oracle.goldengate.datasource.meta.TableMetaData`: represents all metadata for a single table; contains zero or more `ColumnMetaData`.
- `oracle.goldengate.datasource.meta.ColumnMetaData`: contains column names and data types, as defined in the database and/or in the Oracle GoldenGate source definitions file.

See the Javadoc for additional details.

10.2.39.6 Configuring Data Transforms

Data Transforms is the Oracle GoldenGate module for Distributed Applications and Analytics, which can help with column level data transformations during the replicat process.

It's a 2 step process:

1. **Configuring a Matcher:**
Matcher configuration helps in identifying target columns, which you want to apply the Data Transforms on.
 2. **Configuring a Converter:**
Converter defines the logic to be used to convert the matched target columns prior to writing it to the target.
- [Built-in Regex Based Data Transforms](#)
 - [Developing Custom Data Transforms](#)
 - [Additional Configurations](#)
 - [Troubleshooting and Diagnostics](#)

10.2.39.6.1 Built-in Regex Based Data Transforms

By default, Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) provides a default regex based implementation for both matcher and the converter.

Data Transform Configuration

```
# Transform name (To be referred in the subsequent configs)
gg.transforms=t1

# Configure the matcher implementation (using the built-in regex type in this ex)
gg.transform.t1.matcher=regex

# Configure the converter implementation (using the built-in regex type in this ex)
gg.transform.t1.converter=regex

# These matcher configs correspond to the built-in regex matcher

# Target catalogs to match. Default value is *
gg.transform.t1.matcher.catalogRegex={}

# Target schema to match. Default value is *
gg.transform.t1.matcher.schemaRegex={}

# Target tables to match (*Required field)
gg.transform.t1.matcher.tableRegex={}
```

```
# Target columns to match (*Required field)
gg.transform.t1.matcher.columnRegex={}

# These converter configs correspond to the built-in regex converter

# Content search regex (from the columns selected, filter only specific values matching
this regex)
gg.transform.t1.converter.replaceRegex={}

# Content replacement value
gg.transform.t1.converter.replaceString={}
```

Note

tableRegex and columnRegex params do not have any default value. No tables or columns will be matched if either tableRegex or columnRegex is not defined.

Example on how to use the built-in regex based data transform

The following configuration creates a data transform which identifies all the target objects with:

Matcher

1. Table name starting with tab.
2. Column name ending with col.

Converter

1. Converts the above matched column values to a fixed value, for example: TestValue.

```
gg.transforms=t1

gg.transform.t1.matcher=regex
gg.transform.t1.converter=regex

gg.transform.t1.matcher.catalogRegex=. *
gg.transform.t1.matcher.schemaRegex=. *

# Table name starting with 'tab'
gg.transform.t1.matcher.tableRegex=^tab.*

# Column name ending with 'col'
gg.transform.t1.matcher.columnRegex=. *col$

gg.transform.t1.converter.replaceRegex=. *

# Replacement value
gg.transform.t1.converter.replaceString=TestVal
```

10.2.39.6.2 Developing Custom Data Transforms

A custom data transform implementation can be achieved by implementing the matcher and converter interfaces as shown in the example below.

Consider a scenario where you want to mask a sensitive field's value during replicat process.

1. Configure the target column which matches the following criteria:
 - a. **Catalog name:** Cat1
 - b. **Schema name:** Sch1
 - c. **Table name:** Sample_Table
 - d. **Column name:** Sample_Column
2. Configure a converter with some conversion implementation.
 - a. Replace the column values for the above matched column with a masked value

```
@Matcher(id = "matcher1", description = "Custom target column matcher.")
public class CustomTargetMatcher implements TargetMatcher {
    @Override
    public boolean matches(final TableMetaData tableMetaData) {
        return tableMetaData.getCatalogName().equals("Cat1") &&
            tableMetaData.getSchemaName().equals("Sch1") &&
            tableMetaData.getTableName().equals("Sample_Table");
    }
    @Override
    public boolean matches(final ColumnMetaData columnMetaData) {
        return columnMetaData.getColumnName().equals("Sample_Column");
    }
}

@Converter(id = "converter1", description = "Custom data converter.")
public class CustomConverter implements DataConverter {

    public String convert(String originalData, final TableMetaData tableMetaData, final
        ColumnMetaData columnMetaData) {
        return "*****"; // Masked Value
    }
}
```

Adapter properties for this implementation

```
gg.transforms=t1

# This config corresponds to the @Matcher => id param
gg.transform.t1.matcher=matcher1
# This config corresponds to the @Converter => id param
gg.transform.t1.converter=converter1
```

To use the custom classes:

Place the custom classes into a JAR and include them in the classpath. Include the custom JAR files in the JVM classpath using the Java properties (using `java.class.path` in the `jvm.bootoptions` property) or under `gg.classpath`

10.2.39.6.3 Additional Configurations

1. When using data transform for timestamp-based columns, either remove or comment the `replicat` property for default timestamp format `gg.format.timestamp`.
2. By default, column data transform are disabled for Key columns. You can use the following property to enable transform for key columns too: `gg.transform.includeKeyColumn=true`.

Note

In case, there's no Primary Key for a source table, all the table columns comprised together are considered as primary keys. Ensure that the `includeKeyColumn` property is enabled for such cases as well.

10.2.39.6.4 Troubleshooting and Diagnostics

1. Ensure that all the required transform parameters are declared under the replicat properties file.
When the data transform is not configured appropriately and the replicat properties file has missing/invalid `gg.transform` properties, replicat will just skip this transform and continue.

Replicat will also throw the following Warning messages for these scenarios.

`Transform property is not set [gg.transform.{name}.matcher.tableRegex].`
`Transform property is not set [gg.transform.{name}.matcher.columnRegex].`
2. Ensure that the regex specified under each of the matcher/converter properties are valid regex strings.
Replicat will throw the following error message and exception in case there's an invalid regex configured: `PatternSyntaxException` - If the regular expression's syntax is invalid.

Fix the regex errors in order to continue with the replicat process.
3. For the custom transform, ensure the implemented custom class has been correctly added to the classpath.
Replicat throws the following error message in this case and it skips this transform and continues:

`Could not find transform class instance for type {type}.`

Ensure to add the custom class to the `gg.classpath` property.
4. By default, the column data transform is disabled for Key columns. Use the following property to enable transform for key columns: `gg.transform.includeKeyColumn=true`.

Note

In case, there's no Primary Key for a source table, all the table columns comprised together are considered as primary keys. Ensure that the `includeKeyColumn` property is enabled for such cases as well.

Administer

- [Automatic Heartbeat for Oracle GoldenGate for Distributed Applications and Analytics](#)
This article describes how to enable Heartbeat for Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) and how to manage and modify heartbeat across the replication environment.
- [Parsing the Message](#)
- [Message Capture Properties](#)

11.1 Automatic Heartbeat for Oracle GoldenGate for Distributed Applications and Analytics

This article describes how to enable Heartbeat for Oracle GoldenGate for Distributed Applications and Analytics (GG for DAA) and how to manage and modify heartbeat across the replication environment.

- [Overview](#)
Oracle GoldenGate for Distributed Applications and Analytics supports automatic heartbeat tables that monitors the replication lag and connectivity across distributed environments. These heartbeat table records can be processed in two different modes based on your system architecture and target configuration.
- [Automatic Heartbeat Tables](#)

11.1.1 Overview

Oracle GoldenGate for Distributed Applications and Analytics supports automatic heartbeat tables that monitors the replication lag and connectivity across distributed environments. These heartbeat table records can be processed in two different modes based on your system architecture and target configuration.

Prerequisites and Initial Setup

To enable heartbeat table functionality in Oracle GoldenGate for Distributed Applications and Analytics, perform the following steps:

1. Define the Oracle GoldenGate schema in the GLOBALS file.

```
GGSCHEMA GGADMIN
```

2. Enable heartbeat table support by adding the following directive in the GLOBALS file:

```
ENABLE_HEARTBEAT_TABLE
```

3. Apply the changes:

- **For Classic Architecture:**
 - a. Restart the Oracle GoldenGate Software Command Interface (GGSCI).

- b. Restart the Manager.
- **For Microservices Architecture (MA):**
 - a. Stop the Oracle GoldenGate for Distributed Applications and Analytics deployment.
 - b. Restart the Service Manager.
 - c. Start the deployment again.

Mode 1: Passthrough Mode

The Passthrough mode, allows heartbeat table records to be sent directly to non-Oracle targets, such as Kafka, without any transformation or internal processing by Oracle GoldenGate.

Configuration Steps

1. Create the GLOBALS file as outlined in the **Prerequisites and Initial Setup**.

Note

- Do not run the ADD HEARTBEATTABLE command for this mode.
- No need to define MAP statements for heartbeat tables in the Replicat parameter file.

2. Ensure that the GGSHEMA and ENABLE_HEARTBEAT_TABLE directives are correctly set in the GLOBALS file.
3. Restart Oracle GoldenGate services as required based on your architecture.

Mode 2: Heartbeat as User Data

In the Heartbeat as User Data mode, the heartbeat table records are treated like regular user data. The Replicat process writes these records to the **.json** files.

File Locations

Architecture{}	GG_HEARTBEAT File{}	GG_HEARTBEAT_HISTORY File{}
Classic	dirtmp/<replicat_name>-hb.json	dirtmp/<replicat_name>hb-<date>.json
Microservices Architecture	<deployment_home>/var/temp/<replicat_name>-hb.json	<deployment_home>/var/temp/<replicat_name>hb-<date>.json

Configuration Steps

1. Create the GLOBALS file as outlined in the **Prerequisites and Initial Setup**.
2. Run the following Oracle GoldenGate Software Command Interface (GGSCI)Admin Client command to create the heartbeat table.

```
ADD HEARTBEATTABLE
```

Note

No need to define the MAP statements for heartbeat tables in the Replicat parameter file.

3. Ensure that the GGSHEMA and ENABLE_HEARTBEAT_TABLE directives are correctly set in the GLOBALS file.
4. Restart the Oracle GoldenGate services according to the architecture type.

11.1.2 Automatic Heartbeat Tables

- [ADD HEARTBEATTABLE](#)
- [ALTER HEARTBEAT TABLE](#)
- [INFO HEARTBEATTABLE](#)
- [LAG](#)
- [DELETE HEARTBEATTABLE](#)

11.1.2.1 ADD HEARTBEATTABLE

ADD HEARTBEATTABLE

```
[, RETENTION_TIME number in days] |  
[, PURGE_FREQUENCY number in days]
```

RETENTION_TIME

Specifies when heartbeat entries older than the retention time in the history table are purged. The default is 30 days.

PURGE FREQUENCY

Specifies how often the purge scheduler is run to delete table entries that are older than the retention time from the heartbeat history. The default is 1 day.

Example:

```
GGSCI > ADD HEARTBEATTABLE  
HEARTBEAT is now enabled:  
HEARTBEAT configuration file in dirprm\heartbeat.properties  
heartbeat.enabled=true  
heartbeat.frequency=60  
heartbeat.retention_time=30  
heartbeat.purge.frequency=1  
heartbeat.db.name=BigData
```

Note

Ensure to run the ADD HEARTBEATTABLE command before processing the trail file through the replicat.

11.1.2.2 ALTER HEARTBEAT TABLE

ALTER HEARTBEAT TABLE

[, RETENTION_TIME number in days] |

[, PURGE_FREQUENCY number in days]

RETENTION_TIME

Update heartbeat.retention_time in dirprm/heartbeat.properties; will take effect on the next restart.

PURGE_FREQUENCY

Specifies how often entries older than the retention time are purged from the GG_HEARTBEAT_HISTORY. The default is 1 day.

11.1.2.3 INFO HEARTBEAT TABLE

Example

```
HEARTBEAT configuration file dirprm\heartbeat.properties
heartbeat.enabled=true
heartbeat.frequency=60
heartbeat.retention_time=30
heartbeat.purge.frequency=1
heartbeat.db.name=BigData
```

11.1.2.4 LAG

LAG <replicat name>

Example

```
GGSCI> LAG rtpc
Lag Information From Heartbeat Table
LAG          AGE          FROM      TO      PATH
5.77s        10m 22.87s      ORCL      BIGDATA  ETPC ==> PTPC ==> RTPC
```

LAG <replicat name> HISTORY

```
GGSCI> LAG rtpc HISTORY
```

Example

```
Lag Information From Heartbeat Table
LAG      AGE      FROM      TO      PATH
5.77s    10m 22.87s    ORCL      ORCL    ETPC ==> PTPC ==> RTPC

Lag History
DATE      MIN      AVG      MAX
2018-07-01 5.77s    5.90s    6.20s
2018-07-02 6.77s    6.90s    7.20s
2018-07-03 7.77s    7.90s    8.20s
2018-07-04 8.77s    9.90s    9.20s
```

11.1.2.5 DELETE HEARTBEATTABLE

DELETE HEARTBEATTABLE

Example

```
GGSCI> DELETE HEARTBEATTABLE
```

11.2 Parsing the Message

- [Parsing Overview](#)
- [Fixed Width Parsing](#)
- [Delimited Parsing](#)
- [XML Parsing](#)
- [Source Definitions Generation Utility](#)

11.2.1 Parsing Overview

The role of the parser is to translate JMS text message data and header properties into an appropriate set of transactions and operations to pass into the VAM interface. To do this, the parser always must find certain data:

- Transaction identifier
- Sequence identifier
- Timestamp
- Table name
- Operation type
- Column data specific to a particular table name and operation type

Other data will be used if the configuration requires it:

- Transaction indicator
- Transaction name
- Transaction owner

The parser can obtain this data from JMS header properties, system generated values, static values, or in some parser-specific way. This depends on the nature of the piece of information.

- [Parser Types](#)
- [Source and Target Data Definitions](#)
- [Required Data](#)
- [Optional Data](#)

11.2.1.1 Parser Types

The Oracle GoldenGate message capture adapter supports three types of parsers:

- Fixed – Messages contain data presented as fixed width fields in contiguous text.
- Delimited – Messages contain data delimited by field and end of record characters.

- XML – Messages contain XML data accessed through XPath expressions.

11.2.1.2 Source and Target Data Definitions

There are several ways source data definitions can be defined using a combination of properties and external files.

There are several properties that configure how the selected parser gets data and how the source definitions are converted to target definitions.

11.2.1.3 Required Data

The following information is required for the parsers to translate the messages:

- [Transaction Identifier](#)
- [Sequence Identifier](#)
- [Timestamp](#)
- [Table Name](#)
- [Operation Type](#)
- [Column Data](#)

11.2.1.3.1 Transaction Identifier

The transaction identifier (txid) groups operations into transactions as they are written to the Oracle GoldenGate trail file. The Oracle GoldenGate message capture adapter supports only contiguous, non-interleaved transactions. The transaction identifier can be any unique value that increases for each transaction. A system generated value can generally be used.

11.2.1.3.2 Sequence Identifier

The sequence identifier (seqid) identifies each operation internally. This can be used during recovery processing to identify operations that have already been written to the Oracle GoldenGate trail. The sequence identifier can be any unique value that increases for each operation. The length should be fixed.

The JMS Message ID can be used as a sequence identifier if the message identifier for that provider increases and is unique. However, there are cases (for example, using clustering, failed transactions) where JMS does not guarantee message order or when the ID may be unique but not be increasing. The system generated Sequence ID can be used, but it can cause duplicate messages under some recovery situations. The recommended approach is to have the JMS client that adds messages to the queue set the Message ID, a header property, or some data element to an application-generated unique value that is increasing.

11.2.1.3.3 Timestamp

The timestamp (timestamp) is used as the commit timestamp of operations within the Oracle GoldenGate trail. It should be increasing but this is not required, and it does not have to be unique between transactions or operations. It can be any date format that can be parsed.

11.2.1.3.4 Table Name

The table name is used to identify the logical table to which the column data belongs. The adapter requires a two part table name in the form SCHEMA_NAME.TABLE_NAME. This can either

be defined separately (schema and table) or as a combination of schema and table (schemaandtable).

A single field may contain both schema and table name, they may be in separate fields, or the schema may be included in the software code so only the table name is required. How the schema and table names can be specified depends on the parser. In any case the two part logical table name is used to write records in the Oracle GoldenGate trail and to generate the source definitions file that describes the trail.

11.2.1.3.5 Operation Type

The operation type (optype) is used to determine whether an operation is an insert, update or delete when written to the Oracle GoldenGate trail. The operation type value for any specific operation is matched against the values defined for each operation type.

The data written to the Oracle GoldenGate trail for each operation type depends on the Extract configuration:

- Inserts
 - The after values of all columns are written to the trail.
- Updates
 - Default – The after values of keys are written. The after values of columns that have changed are written if the before values are present and can be compared. If before values are not present then all columns are written.
 - NOCOMPRESSUPDATES – The after values of all columns are written to the trail.
 - GETUPDATEBEFORES – The before and after values of columns that have changed are written to the trail if the before values are present and can be compared. If before values are not present only after values are written.
 - If both NOCOMPRESSUPDATES and GETUPDATEBEFORES are included, the before and after values of all columns are written to the trail if before values are present
- Deletes
 - Default – The before values of all keys are written to the trail.
 - NOCOMPRESSDELETES – The before values of all columns are written to the trail.

Primary key update operations may also be generated if the before values of keys are present and do not match the after values.

11.2.1.3.6 Column Data

All parsers retrieve column data from the message text and write it to the Oracle GoldenGate trail. In some cases the columns are read in index order as defined by the source definitions, in other cases they are accessed by name.

Depending on the configuration and original message text, both before and after or only after images of the column data may be available. For updates, the data for non-updated columns may or may not be available.

All column data is retrieved as text. It is converted internally into the correct data type for that column based on the source definitions. Any conversion problem will result in an error and the process will abend.

11.2.1.4 Optional Data

The following data may be included, but is not required.

- [Transaction Indicator](#)
- [Transaction Name](#)
- [Transaction Owner](#)

11.2.1.4.1 Transaction Indicator

The relationship of transactions to messages can be:

- One transaction per message
This is determined automatically by the scope of the message.
- Multiple transactions per message
This is determined by the transaction indicator (txind). If there is no transaction indicator, the XML parser can create transactions based on a matching transaction rule.
- Multiple messages per transaction

The transaction indicator (txind) is required to specify whether the operation is the beginning, middle, end or the whole transaction. The transaction indicator value for any specific operation is matched against the values defined for each transaction indicator type. A transaction is started if the indicator value is beginning or whole, continued if it is middle, and ended if it is end or whole.

11.2.1.4.2 Transaction Name

The transaction name (txname) is optional data that can be used to associate an arbitrary name to a transaction. This can be added to the trail as a token using a GETENV function.

11.2.1.4.3 Transaction Owner

The transaction owner (txowner) is optional data that can be used to associate an arbitrary user name to a transaction. This can be added to the trail as a token using a GETENV function, or used to exclude certain transactions from processing using the EXCLUDEUSER Extract parameter.

11.2.2 Fixed Width Parsing

Fixed width parsing is based on a data definition that defines the position and the length of each field. This is in the format of a Cobol copybook. A set of properties define rules for mapping the copybook to logical records in the Oracle GoldenGate trail and in the source definitions file.

The incoming data should consist of a standard format header followed by a data segment. Both should contain fixed width fields. The data is parsed based on the PIC definition in the copybook. It is written to the trail translated as explained in [Header and Record Data Type Translation](#).

- [Header](#)
- [Header and Record Data Type Translation](#)
- [Key identification](#)

- [Using a Source Definition File](#)

11.2.2.1 Header

The header must be defined by a copybook 01 level record that includes the following:

- A commit timestamp or a change time for the record
- A code to indicate the type of operation: insert, update, or delete
- The copybook record name to use when parsing the data segment

Any fields in the header record that are not mapped to Oracle GoldenGate header fields are output as columns.

The following example shows a copybook definition containing the required header values

Example 11-1 Specifying a Header

```
01 HEADER.  
20 Hdr-Timestamp          PIC X(23)  
20 Hdr-Source-DB-Function PIC X  
20 Hdr-Source-DB-Rec-ID   PIC X(8)
```

For the preceding example, you must set the following properties:

```
fixed.header=HEADER  
fixed.timestamp=Hdr-Timestamp  
fixed.optype=Hdr-Source-DB-Function  
fixed.table=Hdr-Source-DB-Rec-Id
```

The logical name table output in this case will be the value of Hdr-Source-DB-Rec-Id.

- [Specifying Compound Table Names](#)
- [Specifying timestamp Formats](#)
- [Specifying the Function](#)

11.2.2.1.1 Specifying Compound Table Names

More than one field can be used for a table name. For example, you can define the logical schema name through a static property such as:

```
fixed.schema=MYSHEMA
```

You can then add a property that defines the data record as multiple fields from the copybook header definition.

Example 11-2 Specifying Compound Table Names

```
01 HEADER.  
20 Hdr-Source-DB          PIC X(8).  
20 Hdr-Source-DB-Rec-Id   PIC X(8).  
20 Hdr-Source-DB-Rec-Version PIC 9(4).  
20 Hdr-Source-DB-Function PIC X.  
20 Hdr-Timestamp          PIC X(22).
```

For the preceding example, you must set the following properties:

```
fixed.header=HEADER  
fixed.table=Hdr-Source-DB-Rec-Id,Hdr-Source-DB-Rec-Version  
fixed.schema=MYSHEMA
```

The fields will be concatenated to result in logical schema and table names of the form:

MYSHEMA.Hdr-Source-DB-Rec-Id+Hdr-Source-DB-Rec-Version

11.2.2.1.2 Specifying timestamp Formats

A timestamp is parsed using the default format YYYY-MM-DD HH:MM:SS.FFF, with FFF depending on the size of the field.

Specify different incoming formats by entering a comment before the datetime field as shown in the next example.

Example 11-3 Specifying timestamp formats

```
01 HEADER.
* DATEFORMAT YYYY-MM-DD-HH.MM.SS.FF
  20 Hdr-Timestamp      PIC X(23)
```

11.2.2.1.3 Specifying the Function

Use properties to map the standard Oracle GoldenGate operation types to the optype values. The following example specifies that the operation type is in the Hdr-Source-DB-Function field and that the value for insert is A, update is U and delete is D.

Example 11-4 Specifying the Function

```
fixed.optype=Hdr-Source-DB-Function
fixed.optype.insert=A
fixed.optype.update=U
fixed.optype.delete=D
```

11.2.2.2 Header and Record Data Type Translation

The data in the header and the record data are written to the trail based on the translated data type.

- A field definition preceded by a date format comment is translated to an Oracle GoldenGate datetime field of the specified size. If there is no date format comment, the field will be defined by its underlying data type.
- A PIC X field is translated to the CHAR data type of the indicated size.
- A PIC 9 field is translated to a NUMBER data type with the defined precision and scale. Numbers that are signed or unsigned and those with or without decimals are supported.

The following examples show the translation for various PIC definitions.

Input	Output
PIC XX	CHAR(2)
PIC X(16)	CHAR(16)
PIC 9(4)	NUMBER(4)
* YYMMDD PIC 9(6)	DATE(10) YYYY-MM-DD

Input	Output
PIC 99.99	NUMBER(4,2)

In the example an input YYMMDD date of 100522 is translated to 2010-05-22. The number 1234567 with the specified format PIC 9(5)V99 is translated to a seven digit number with two decimal places, or 12345.67.

11.2.2.3 Key identification

A comment is used to identify key columns within the data record.

In the following example Account has been marked as a key column for TABLE1.

```
01 TABLE1
* KEY
20 Account      PIC X(19)
20 PAN_Seq_Num PIC 9(3)
```

11.2.2.4 Using a Source Definition File

You can use fixed width parsing based on a data definition that comes from an Oracle GoldenGate source definition file. This is similar to Cobol copybook because a source definition file contains the position and the length of each field of participating tables. To use a source definition file, you must set the following properties:

```
fixed.userdefs.tables=qasource.HEADER
fixed.userdefs.qasource.HEADER.columns=optype, schemaandtable
fixed.userdefs.qasource.HEADER.optype=vchar 3
fixed.userdefs.qasource.HEADER.schemaandtable=vchar 30
```

```
fixed.header=qasource.HEADER
```

The following example defines a header section of a total length of 33 characters; the first 3 characters are the operation type, and the last 30 characters is the table name. The layout of all records to be parsed must start with the complete header section as defined in the fixed.userdefs properties. For each record, the header section is immediately followed by the content of all column data for the corresponding table. The column data must be strictly laid out according to its offset and length defined in the source definition file. Specifically, the offset information is the fourth field (Fetch Offset) of the column definition and the length information is the third field (External Length) of the column definition. The following is an example of a definition for GG.JMSCAP_TCUSTMER:

```
Definition for table GG.JMSCAP_TCUSTMER
```

```
Record length: 78
```

```
Syskey: 0
```

```
Columns: 4
```

```
CUST_CODE  64      4      0 0 0 1 0      4      4      0 0 0 0 0 1      0
1 0
NAME       64     30     10 0 0 1 0     30     30     0 0 0 0 0 1      0
0 0
CITY       64     20     46 0 0 1 0     20     20     0 0 0 0 0 1      0
0 0
```



```
STATE      0      2      72  0  0  1  0      2      2      0  0  0  0  0  1      0
0 0
End of definition
```

The fixed width data for GG.JMSCAP_TCUSTMER may be similar to the following where the offset guides have been added to each section for clarity:

```
0      1      2      3  0      1      2      3
4      5      6      7      8
01234567890123456789012345678901201234567890123456789012345678901234
567890123456789012345678901234567890
I  GG.JMSCAP_TCUSTMER      WILL      BG SOFTWARE
CO.      SEATTLE      WA
I  GG.JMSCAP_TCUSTMER      JANE      ROCKY FLYER
INC.      DENVER      CO
I  GG.JMSCAP_TCUSTMER      DAVE      DAVE'S PLANES
INC.      TALLAHASSEE      FL
I  GG.JMSCAP_TCUSTMER      BILL      BILL'S USED
CARS      DENVER      CO
I  GG.JMSCAP_TCUSTMER      ANN      ANN'S
BOATS      SEATTLE      WA
U  GG.JMSCAP_TCUSTMER      ANN      ANN'S
BOATS      NEW YORK      NY
```

You can choose to specify shorter data records, which means that only some of the earlier columns are present. To do this, the following requirements must be met:

- None of the missing or omitted columns are part of the key and
- all columns that are present contain complete data according to their respective External Length information

11.2.3 Delimited Parsing

Delimited parsing is based a preexisting source definitions files and a set of properties. The properties specify the delimiters to use and other rules, such as whether there are column names and before values. The source definitions file determines the valid tables to be processed and the order and data type of the columns in the tables.

The format of the delimited message is:

```
METACOLSn[, COLNAMES]m[, COLBEFOREVALS]m, {COLVALUES}m\n
```

Where:

- There can be *n* metadata columns each followed by a field delimiter such as the comma shown in the format statement.
- There can be *m* column values. Each of these are preceded by a field delimiter such as a comma.
- The column name and before value are optional.
- Each record is terminated by an end of line delimiter, such as \n.

The message to be parsed *must* contain at least the header and metadata columns. If the number of columns is fewer than the number of header and meta columns, then the capture process terminates and provides an error message.

The remaining number of columns after the header and metadata columns are the column data for the corresponding table, specified in the order of the columns in the resolved metadata. Ideally, the number of table columns present in the message is exactly the same as the expected number of columns according to the metadata. However, missing columns in the message towards the end of message is allowed and the parser marks those last columns (not present in the rest of the message) as missing column data.

Although missing data is allowed from parser perspective, if the key @ column(s) is/are missing, then the capture process will also terminate.

Oracle GoldenGate primary key updates and unified updates are not supported. The only supported operations are inserts, updates, deletes, and truncates.

- [Metadata Columns](#)
- [Parsing Properties](#)
- [Parsing Steps](#)

11.2.3.1 Metadata Columns

The metadata columns correspond to the header and contain fields that have special meaning. Metadata columns should include the following information.

- **optype** contains values indicating if the record is an insert, update, or delete. The default values are I, U, and D.
- **timestamp** indicates type of value to use for the commit timestamp of the record. The format of the timestamp defaults to YYYY-DD-MM HH:MM:SS.FFF.
- **schemaandtable** is the full table name for the record in the format SCHEMA.TABLE.
- **schema** is the record's schema name.
- **table** is the record's table name.
- **txind** is a value that indicates whether the record is the beginning, middle, end or the only record in the transaction. The default values are 0, 1, 2, 3.
- **id** is the value used as the sequence number (RSN or CSN) of the record. The id of the first record (operation) in the transaction is used for the sequence number of the transaction.

11.2.3.2 Parsing Properties

Properties can be set to describe delimiters, values, and date and time formats.

- [Properties to Describe Delimiters](#)
- [Properties to Describe Values](#)
- [Properties to Describe Date and Time](#)

11.2.3.2.1 Properties to Describe Delimiters

The following properties determine the parsing rules for delimiting the record.

- **fielddelim** specifies one or more ASCII or hexadecimal characters as the value for the field delimiter

- **recorddelim** specifies one or more ASCII or hexadecimal characters as the value for the record delimiter
- **quote** specifies one or more ASCII or hexadecimal characters to use for quoted values
- **nullindicator** specifies one or more ASCII or hexadecimal characters to use for NULL values

You can define escape characters for the delimiters so they will be replaced if the characters are found in the text. For example if a backslash and apostrophe (\') are specified, then the input "They used Mike\'s truck" is translated to "They used Mike's truck". Or if two quotes (") are specified, "They call him ""Big Al"" is translated to "They call him "Big Al"".

Data values may be present in the record without quotes, but the system only removes escape characters within quoted values. A non-quoted string that matches a null indicator is treated as null.

11.2.3.2.2 Properties to Describe Values

The following properties provide more information:

- **hasbefore** indicates before values are present for each record
- **hasnames** indicates column names are present for each record
- **afterfirst** indicates column after values come before column before values
- **isgrouped** indicates all column names, before values and after values are grouped together in three blocks, rather than alternately per column

11.2.3.2.3 Properties to Describe Date and Time

The default format YYYY-DD-MM HH:MM:SS.FFF is used to parse dates. You can use properties to override this on a global, table or column level. Examples of changing the format are shown below.

```
delim.dateformat.default=MM/DD/YYYY-HH:MM:SS  
delim.dateformat.MY.TABLE=DD/MMM/YYYY  
delim.dateformat.MY.TABLE.COL1=MMYYYY
```

11.2.3.3 Parsing Steps

The steps in delimited parsing are:

1. The parser first reads and validates the metadata columns for each record.
2. This provides the table name, which can then be used to look up column definitions for that table in the source definitions file.
3. If a definition cannot be found for a table, the processing will stop.
4. Otherwise the columns are parsed and output to the trail in the order and format defined by the source definitions.

11.2.4 XML Parsing

XML parsing is based on a preexisting source definitions file and a set of properties. The properties specify rules to determine XML elements and attributes that correspond to transactions, operations and columns. The source definitions file determines the valid tables to be processed and the ordering and data types of columns in those tables.

- [Styles of XML](#)

- [XML Parsing Rules](#)
- [XPath Expressions](#)
- [Other Value Expressions](#)
- [Transaction Rules](#)
- [Operation Rules](#)
- [Column Rules](#)
- [Overall Rules Example](#)

11.2.4.1 Styles of XML

The XML message is formatted in either dynamic or static XML. At runtime the contents of dynamic XML are data values that cannot be predetermined using a sample XML or XSD document. The contents of static XML that determine tables and column element or attribute names can be predetermined using those sample documents.

The following two examples contain the same data.

Example 11-5 An Example of Static XML

```
<NewMyTableEntries>
  <NewMyTableEntry>
    <CreateTime>2010-02-05:10:11:21</CreateTime>
    <KeyCol>keyval</KeyCol>
    <Col1>col1val</Col1>
  </NewMyTableEntry>
</NewMyTableEntries>
```

The NewMyTableEntries element marks the transaction boundaries. The NewMyTableEntry indicates an insert to MY.TABLE. The timestamp is present in an element text value, and the column names are indicated by element names.

You can define rules in the properties file to parse either of these two styles of XML through a set of XPath-like properties. The goal of the properties is to map the XML to a predefined source definitions file through XPath matches.

Example 11-6 An Example of Dynamic XML

```
<transaction id="1234" ts="2010-02-05:10:11:21">
  <operation table="MY.TABLE" optype="I">
    <column name="keycol" index="0">
      <aftervalue><![CDATA[keyval]]></aftervalue>
    </column>
    <column name="col1" index="1">
      <aftervalue><![CDATA[col1val]]></aftervalue>
    </column>
  </operation>
</transaction>
```

Every operation to every table has the same basic message structure consisting of transaction, operation and column elements. The table name, operation type, timestamp, column names, column values, etc. are obtained from attribute or element text values.

11.2.4.2 XML Parsing Rules

Independent of the style of XML, the parsing process needs to determine:

- Transaction boundaries

- Operation entries and metadata including:
 - Table name
 - Operation type
 - Timestamp
- Column entries and metadata including:
 - Either the column name or index; if both are specified the system will check to see if the column with the specified data has the specified name.
 - Column before or after values, sometimes both.

This is done through a set of interrelated rules. For each type of XML message that is to be processed you name a rule that will be used to obtain the required data. For each of these named rules you add properties to:

- Specify the rule as a transaction, operation, or column rule type. Rules of any type are required to have a specified name and type.
- Specify the XPath expression to match to see if the rule is active for the document being processed. This is optional; if not defined the parser will match the node of the parent rule or the whole document if this is the first rule.
- List detailed rules (subrules) that are to be processed in the order listed. Which subrules are valid is determined by the rule type. Subrules are optional.

In the following example the top-level rule is defined as genericrule. It is a transaction type rule. Its subrules are defined in oprule and they are of the type operation.

```
xmlparser.rules=genericrule
xmlparser.rules.genericrule.type=tx
xmlparser.rules.genericrule.subrules=oprule
xmlparser.rules.oprule.type=op
```

11.2.4.3 XPath Expressions

The XML parser supports a subset of XPath expressions necessary to match elements and Extract data. An expression can be used to match a particular element or to Extract data.

When doing data extraction most of the path is used to match. The tail of the expression is used for extraction.

- [Supported Constructs:](#)
- [Supported Expressions](#)
- [Obtaining Data Values](#)

11.2.4.3.1 Supported Constructs:

Supported Constructs	Description
/e	Use the absolute path from the root of the document to match e.
./e or e	Use the relative path from current node being processed to match e.
../e	Use a path based on the parent of the current node (can be repeated) to match e.

Supported Constructs	Description
//e	Match e wherever it occurs in a document.
*	Match any element. Note: Partially wild-carded names are not supported.
[n]	Match the nth occurrence of an expression.
[x=v]	Match when x is equal to some value v where x can be: • @att - some attribute value • text() - some text value • name() - some name value • position() - the element position

11.2.4.3.2 Supported Expressions

Supported Expressions	Descriptions
Match root element	/My/Element
Match sub element to current node	./Sub/Element
Match nth element	/My/*[n]
Match nth Some element	/My/Some[n]
Match any text value	/My/*[text() = 'value']
Match the text in Some element	/My/Some[text() = 'value']
Match any attribute	/My/*[@att = 'value']
Match the attribute in Some element	/My/Some[@att = 'value']

11.2.4.3.3 Obtaining Data Values

In addition to matching paths, the XPath expressions can also be used to obtain data values, either absolutely or relative to the current node being processed. Data value expressions can contain any of the path elements in the preceding table, but must end with one of the value accessors listed below.

Value Accessors	Description
@att	Some attribute value.
text()	The text content (value) of an element.

Value Accessors	Description
<code>content()</code>	The full content of an element, including any child XML nodes.
<code>name()</code>	The name of an element.
<code>position()</code>	The position of an element in its parent.

Example 11-7 Examples of Extracting Data Values

To extract the relative element text value:

```
/My/Element/text()
```

To extract the absolute attribute value:

```
/My/Element/@att
```

To extract element text value with a match:

```
/My/Some[@att = 'value']/Sub/text()
```

Note

Path accessors, such as ancestor/descendent/self, are not supported.

11.2.4.4 Other Value Expressions

The values extracted by the XML parser are either column values or properties of the transaction or operation, such as table or timestamp. These values are either obtained from XML using XPath or through properties of the JMS message, system values, or hard coded values. The XML parser properties specify which of these options are valid for obtaining the values for that property.

The following example specifies that timestamp can be an XPath expression, a JMS property, or the system generated timestamp.

```
{txrule}.timestamp={xpath-expression}|${jms-property}|*ts
```

The next example specifies that table can be an XPath expression, a JMS property, or hard coded value.

```
{oprule}.table={xpath-expression}|${jms-property}|"value"
```

The last example specifies that name can be a XPath expression or hard coded value.

```
{colrule}.timestamp={xpath-expression}|"value"
```

11.2.4.5 Transaction Rules

The rule that specifies the boundary for a transaction is at the highest level. Messages may contain a single transaction, multiple transactions, or a part of a transaction that spans messages. These are specified as follows:

- **single** - The transaction rule match is not defined.
- **multiple** - Each transaction rule match defines new transaction.
- **span** – No transaction rule is defined; instead a transaction indicator is specified in an operation rule.

For a transaction rule, the following properties of the rule may also be defined through XPath or other expressions:

- **timestamp** – The time at which the transaction occurred.
- **txid** – The identifier for the transaction.

Transaction rules can have multiple subrules, but each must be of type operation.

The following example specifies a transaction that is the whole message and includes a timestamp that comes from the JMS property.

Example 11-8 JMS Timestamp

```
singletxrule.timestamp=$JMSTimeStamp
```

The following example matches the root element transaction and obtains the timestamp from the ts attribute.

Example 11-9 ts Timestamp

```
dyntxrule.match=/Transaction
dyntxrule.timestamp=@ts
```

11.2.4.6 Operation Rules

An operation rule can either be a sub rule of a transaction rule, or a highest level rule (if the transaction is a property of the operation).

In addition to the standard rule properties, an operation rule should also define the following through XPath or other expressions:

- **timestamp** – The timestamp of the operation. This is optional if the transaction rule is defined.
- **table** – The name of the table on which this is an operation. Use this with schema.
- **schema** – The name of schema for the table.
- **schemaandtable** – Both schema and table name together in the form SCHEMA.TABLE. This can be used in place of the individual table and schema properties.
- **optype** – Specifies whether this is an insert, update or delete operation based on optype values:
 - **optype.insertval** – The value indicating an insert. The default is I.
 - **optype.updateval** – The value indicating an update. The default is U.
 - **optype.deleteval** – The value indicating a delete. The default is D.
- **seqid** – The identifier for the operation. This will be the transaction identifier if txid has not already been defined at the transaction level.
- **txind** – Specifies whether this operation is the beginning of a transaction, in the middle or at the end; or if it is the whole operation. This property is optional and not valid if the operation rule is a sub rule of a transaction rule.

Operation rules can have multiple sub rules of type operation or column.

The following example dynamically obtains operation information from the `/Operation` element of a `/Transaction`.

Example 11-10 Operation

```
dynoprule.match= ./Operation
dynoprule.schemaandtable=@table
dynoprule.optype=@type
```

The following example statically matches `/NewMyTableEntry` element to an insert operation on the `MY.TABLE` table.

Example 11-11 Operation example

```
statoprule.match= ./NewMyTableEntry
statoprule.schemaandtable="MY.TABLE"
statoprule.optype="I"
statoprule.timestamp= ./CreateTime/text()
```

11.2.4.7 Column Rules

A column rule must be a sub rule of an operation rule. In addition to the standard rule properties, a column rule should also define the following through XPath or other expressions.

- **name** – The name of the column within the table definition.
- **index** – The index of the column within the table definition.

Note

If only one of `name` and `index` is defined, the other will be determined.

- **before.value** – The before value of the column. This is required for deletes, but is optional for updates.
- **before.isnull** – Indicates whether the before value of the column is null.
- **before.ismissing** – Indicates whether the before value of the column is missing.
- **after.value** – The before value of the column. This is required for deletes, but is optional for updates.
- **after.isnull** – Indicates whether the before value of the column is null.
- **after.ismissing** – Indicates whether the before value of the column is missing.
- **value** – An expression to use for both `before.value` and `after.value` unless overridden by specific before or after values. Note that this does not support different before values for updates.
- **isnull** – An expression to use for both `before.isnull` and `after.isnull` unless overridden.
- **ismissing** – An expression to use for both `before.ismissing` and `after.ismissing` unless overridden.

Dynamic Extraction of Column Information

The following example dynamically obtains column information from the `/Column` element of an `/Operation`

```
dyncolrule.match= ./Column
dyncolrule.name=@name
```

```
dyncolrule.before.value=../beforevalue/text()
dyncolrule.after.value=../aftervalue/text()
```

Static Matching of Elements to Columns

The following example statically matches the /KeyCol and /Col1 elements to columns in MY.TABLE.

```
statkeycolrule.match=/KeyCol
statkeycolrule.name="keycol"
statkeycolrule.value=../text()
statcol1rule.match=/Col1
statcol1rule.name="col1"
statcol1rule.value=../text()
```

11.2.4.8 Overall Rules Example

The following example uses the XML samples shown earlier with appropriate rules to generate the same resulting operation on the MY.TABLE table.

Dynamic XML	Static XML
<pre><transaction id="1234" ts="2010-02-05:10:11:21"> <operation table="MY.TABLE" optype="I"> <column name="keycol" index="0"> <aftervalue> <![CDATA[keyval]]> </aftervalue> </column> <column name="col1" index="1"> <aftervalue> <![CDATA[col1val]]> </aftervalue> </column> </operation> </transaction></pre>	<pre>NewMyTableEntries> <NewMyTableEntry> <CreateTime> 2010-02-05:10:11:21 </CreateTime> <KeyCol>keyval</KeyCol> <Col1>col1val</Col1> </NewMyTableEntry> </NewMyTableEntries></pre>

Dynamic	Static
<pre>dyntxrule.match=/Transaction dyntxrule.timestamp=@ts dyntxrule.subrules=dynoprule dynoprule.match=../Operation dynoprule.schemaandtable=@table dynoprule.optype=@type dynoprule.subrules=dyncolrule dyncolrule.match=../Column dyncolrule.name=@name</pre>	<pre>stattxrule.match=/NewMyTableEntries stattxrule.subrules= statoprule statoprule.match=../NewMyTableEntry statoprule.schemaandtable="MY.TABLE" statoprule.optype="I" statoprule.timestamp=../CreateTime/text() statoprule.subrules= statkeycolrule, statcol1rule statkeycolrule.match=/KeyCol</pre>

```
INSERT INTO MY.TABLE (KEYCOL, COL1)
VALUES ('keyval', 'col1val')
```

11.2.5 Source Definitions Generation Utility

By default, the JMS capture process writes metadata information in the produced trail files, allowing trail file consumers to understand the structure of the trail records without any help from an external definition file.

The output source definitions file can then be used in a pump or delivery process to interpret the trail data created through the VAM.

11.3 Message Capture Properties

- [Logging and Connection Properties](#)
- [Parser Properties](#)

11.3.1 Logging and Connection Properties

The following properties control the connection to JMS and the log file names, error handling, and message output.

- [Logging Properties](#)
- [JMS Connection Properties](#)
- [JNDI Properties](#)

11.3.1.1 Logging Properties

Logging is controlled by the following properties.

- [gg.log](#)
- [gg.log.level](#)
- [gg.log.file](#)
- [gg.log.classpath](#)

11.3.1.1.1 gg.log

Specifies the type of logging that is to be used. The default implementation is the JDK option. This is the built-in Java logging called `java.util.logging (JUL)`. The other logging options are `log4j` or `logback`. The syntax is:

```
gg.log={JDK|log4j|logback}
```

For example, to set the type of logging to `log4j`:

```
gg.log=log4j
```

The log file is created in the report subdirectory of the installation. The default log file name includes the group name of the associated Extract and the file extension is `log`.

11.3.1.1.2 gg.log.level

Specifies the overall log level for all modules. The syntax is:

```
gg.log.level={ERROR|WARN|INFO|DEBUG}
```

The log levels are defined as follows:

- ERROR – Only write messages if errors occur
- WARN – Write error and warning messages
- INFO – Write error, warning and informational messages
- DEBUG – Write all messages, including debug ones.

The default logging level is INFO. The messages in this case will be produced on startup, shutdown and periodically during operation. If the level is switched to DEBUG, large volumes of messages may occur which could impact performance. For example, the following sets the global logging level to INFO:

```
# global logging level  
gg.log.level=INFO
```

11.3.1.1.3 gg.log.file

Specifies the path to the log file. The syntax is:

```
gg.log.file=path_to_file
```

Where the *path_to_file* is the fully defined location of the log file. This allows a change to the name of the log, but you must include the Replicat name if you have more than one Replicat to avoid one overwriting the log of the other.

11.3.1.1.4 gg.log.classpath

Specifies the classpath to the JARs used to implement logging.

```
gg.log.classpath=path_to_jars
```

11.3.1.2 JMS Connection Properties

The JMS connection properties set up the connection, such as how to start up the JVM for JMS integration.

- [jvm.boot.options](#)
- [jms.report.output](#)
- [jms.report.time](#)
- [jms.report.records](#)
- [jms.id](#)
- [jms.destination](#)
- [jms.connectionFactory](#)
- [jms.user, jms.password](#)

11.3.1.2.1 jvm.boot.options

Specifies the classpath and boot options that will be applied when the JVM starts up. The path needs colon (:) separators for UNIX/Linux and semicolons (;) for Windows.

The syntax is:

```
jvm.bootoptions=option[, option][. . .]
```

The *options* are the same as those passed to Java executed from the command line. They may include classpath, system properties, and JVM memory options (such as maximum memory or initial memory) that are valid for the version of Java being used. Valid options may vary based on the JVM version and provider.

For example (all on a single line):

```
jvm.bootoptions= -Djava.class.path=ggjava/ggjava.jar  
-Dlog4j.configuration=my-log4j.properties
```

The `log4j.configuration` property could be a fully qualified URL to a log4j properties file; by default this file is searched for in the classpath. You may use your own log4j configuration, or one of the pre-configured log4j settings: `log4j.properties` (default level of logging), `debug-log4j.properties` (debug logging) or `trace-log4j.properties` (very verbose logging).

11.3.1.2.2 jms.report.output

Specifies where the JMS report is written. The syntax is:

```
jms.report.output={report|log|both}
```

Where:

- `report` sends the JMS report to the Oracle GoldenGate report file. This is the default.
- `log` will write to the Java log file (if one is configured)
- `both` will send to both locations.

11.3.1.2.3 jms.report.time

Specifies the frequency of report generation based on time.

```
jms.report.time=time_specification
```

The following examples write a report every 30 seconds, 45 minutes and eight hours.

```
jms.report.time=30sec  
jms.report.time=45min  
jms.report.time=8hr
```

11.3.1.2.4 jms.report.records

Specifies the frequency of report generation based on number of records. The syntax is:

```
jms.report.records=number
```

The following example writes a report every 1000 records.

```
jms.report.records=1000
```

11.3.1.2.5 jms.id

Specifies that a unique identifier with the indicated format is passed back from the JMS integration to the message capture VAM. This may be used by the VAM as a unique sequence ID for records.

```
jms.id={ogg|time|wmq|activemq|message_header|custom_java_class}
```

Where:

- `ogg` - returns the message header property `GG_ID` which is set by Oracle GoldenGate JMS delivery.
- `time` - uses a system timestamp as a starting point for the message ID
- `wmq` - reformats a WebSphere MQ Message ID for use with the VAM
- `activemq` - reformats an ActiveMQ Message ID for use with the VAM
- `message_header` - specifies your customized JMS message header to be included, such as `JMSMessageID`, `JMSCorrelationID`, or `JMSTimestamp`.
- `custom_java_class` - specifies a custom Java class that creates a string to be used as an ID.

For example:

```
jms.id=time  
jms.id=JMSMessageID
```

The ID returned must be unique, incrementing, and fixed-width. If there are duplicate numbers, the duplicates are skipped. If the message ID changes length, the Extract process will abend.

11.3.1.2.6 `jms.destination`

Specifies the queue or topic name to be looked up using JNDI.

```
jms.destination=jndi_name
```

For example:

```
jms.destination=sampleQ
```

11.3.1.2.7 `jms.connectionFactory`

Specifies the connection factory name to be looked up using JNDI.

```
jms.connectionFactory=jndi_name
```

For example

```
jms.connectionFactory=ConnectionFactory
```

11.3.1.2.8 `jms.user`, `jms.password`

Sets the user name and password of the JMS connection, as specified by the JMS provider.

```
jms.user=user_name  
jms.password=password
```

This is not used for JNDI security. To set JNDI authentication, see the JNDI `java.naming.security` properties.

For example:

```
jms.user=myuser  
jms.password=mypasswd
```

11.3.1.3 JNDI Properties

In addition to specific properties for the message capture VAM, the JMS integration also supports setting JNDI properties required for connection to an Initial Context to look up the connection factory and destination. The following properties must be set:

```
java.naming.provider.url=url  
java.naming.factory.initial=java_class_name
```

If JNDI security is enabled, the following properties may be set:

```
java.naming.security.principal=user_name  
java.naming.security.credentials=password_or_other_authenticator
```

For example:

```
java.naming.provider.url= t3://localhost:7001  
java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory  
java.naming.security.principal=jndiuser  
java.naming.security.credentials=jndipw
```

11.3.2 Parser Properties

Properties specify the formats of the message and the translation rules for each type of parser: fixed, delimited, or XML. Set the `parser.type` property to specify which parser to use. The remaining properties are parser specific.

- [Setting the Type of Parser](#)
- [Fixed Parser Properties](#)
- [Delimited Parser Properties](#)
- [XML Parser Properties](#)

11.3.2.1 Setting the Type of Parser

The following property sets the parser type.

- [parser.type](#)

11.3.2.1.1 parser.type

Specifies the parser to use.

```
parser.type={fixed|delim|xml}
```

Where:

- `fixed` invokes the fixed width parser
- `delim` invokes the delimited parser
- `xml` invokes the XML parser

For example:

```
parser.type=delim
```

11.3.2.2 Fixed Parser Properties

The following properties are required for the fixed parser.

- [fixed.schematype](#)
- [fixed.sourcedefs](#)
- [fixed.copybook](#)

- [fixed.header](#)
- [fixed.seqid](#)
- [fixed.timestamp](#)
- [fixed.timestamp.format](#)
- [fixed.txid](#)
- [fixed.txowner](#)
- [fixed.txname](#)
- [fixed.optype](#)
- [fixed.optype.insertval](#)
- [fixed.optype.updateval](#)
- [fixed.optype.deleteval](#)
- [fixed.table](#)
- [fixed.schema](#)
- [fixed.txind](#)
- [fixed.txind.beginval](#)
- [fixed.txind.middleval](#)
- [fixed.txind.endval](#)
- [fixed.txind.wholeval](#)

11.3.2.2.1 fixed.schematype

Specifies the type of file used as metadata for message capture. The two valid options are `sourcedefs` and `copybook`.

```
fixed.schematype={sourcedefs|copybook}
```

For example:

```
fixed.schematype=copybook
```

The value of this property determines the other properties that must be set in order to successfully parse the incoming data.

11.3.2.2.2 fixed.sourcedefs

If the `fixed.schematype=sourcedefs`, this property specifies the location of the source definitions file that is to be used.

```
fixed.sourcedefs=file_location
```

For example:

```
fixed.sourcedefs=dirdef/hrdemo.def
```

11.3.2.2.3 fixed.copybook

If the `fixed.schematype=copybook`, this property specifies the location of the copybook file to be used by the message capture process.

```
fixed.copybook=file_location
```


For example:

```
fixed.copybook=test_copy_book.cpy
```

11.3.2.2.4 fixed.header

Specifies the name of the sourcedefs entry or copybook record that contains header information used to determine the data block structure:

```
fixed.header=record_name
```

For example:

```
fixed.header=HEADER
```

11.3.2.2.5 fixed.seqid

Specifies the name of the header field, JMS property, or system value that contains the seqid used to uniquely identify individual records. This value must be continually incrementing and the last character must be the least significant.

```
fixed.seqid={field_name|jms_property|*seqid}
```

Where:

- `field_name` indicates the name of a header field containing the `seqid`
- `jms_property` uses the value of the specified JMS header property. A special value of this is `$jmsid` which uses the value returned by the mechanism chosen by the `jms.id` property
- `seqid` indicates a simple incrementing 64-bit integer generated by the system

For example:

```
fixed.seqid=$jmsid
```

11.3.2.2.6 fixed.timestamp

Specifies the name of the field, JMS property, or system value that contains the timestamp.

```
fixed.timestamp={field_name|jms_property|*ts}
```

For example:

```
fixed.timestamp=TIMESTAMP  
fixed.timestamp=$JMSTimeStamp  
fixed.timestamp=*ts
```

11.3.2.2.7 fixed.timestamp.format

Specifies the format of the timestamp field.

```
fixed.timestamp.format=format
```

Where the format can include punctuation characters plus:

- `YYYY` – four digit year
- `YY` – two digit year
- `M[M]` – one or two digit month
- `D[D]` – one or two digit day

- HH – hours in twenty four hour notation
- MI – minutes
- SS – seconds
- Fn – n number of fractions

The default format is "YYYY-MM-DD:HH:MI:SS.FFF"

For example:

```
fixed.timestamp.format=YYYY-MM-DD-HH.MI.SS
```

11.3.2.2.8 fixed.txid

Specifies the name of the field, JMS property, or system value that contains the txid used to uniquely identify transactions. This value must increment for each transaction.

```
fixed.txid={field_name|$jms_property|*txid}
```

For most cases using the system value of *txid is preferred.

For example:

```
fixed.txid=$JMSTxId  
fixed.txid=*txid
```

11.3.2.2.9 fixed.txowner

Specifies the name of the field, JMS property, or static value that contains a user name associated with a transaction. This value may be used to exclude certain transactions from processing. This is an optional property.

```
fixed.txowner={field_name|$jms_property|"value"}
```

For example:

```
fixed.txowner=$MessageOwner  
fixed.txowner="jsmith"
```

11.3.2.2.10 fixed.txname

Specifies the name of the field, JMS property, or static value that contains an arbitrary name to be associated with a transaction. This is an optional property.

```
fixed.txname={field_name|$jms_property|"value"}
```

For example:

```
fixed.txname="fixedtx"
```

11.3.2.2.11 fixed.optype

Specifies the name of the field, or JMS property that contains the operation type, which is validated against the fixed.optype values specified in the next sections.

```
fixed.header.optype={field_name|$jms_property}
```

For example:

```
fixed.header.optype=FUNCTION
```

11.3.2.2.12 fixed.optype.insertval

This value identifies an insert operation. The default is I.

`fixed.optype.insertval={value|\xhex_value}`

For example:

`fixed.optype.insertval=A`

11.3.2.2.13 fixed.optype.updateval

This value identifies an update operation. The default is U.

`fixed.optype.updateval={value|\xhex_value}`

For example:

`fixed.optype.updateval=M`

11.3.2.2.14 fixed.optype.deleteval

This value identifies a delete operation. The default is D.

`fixed.optype.deleteval={value|\xhex_value}`

For example:

`fixed.optype.deleteval=R`

11.3.2.2.15 fixed.table

Specifies the name of the table. This enables the parser to find the data record definition needed to translate the non-header data portion.

`fixed.table=field_name|$jms_property[, . . .]`

More than one comma delimited field name may be used to determine the name of the table. Each field name corresponds to a field in the header record defined by the `fixed.header` property or JMS property. The values of these fields are concatenated to identify the data record.

For example:

`fixed.table=$JMSTableName`
`fixed.table=SOURCE_Db,SOURCE_Db_Rec_Version`

11.3.2.2.16 fixed.schema

Specifies the static name of the schema when generating SCHEMA.TABLE table names.

`fixed.schema="value"`

For example:

`fixed.schema="OGG"`

11.3.2.2.17 fixed.txind

Specifies the name of the field or JMS property that contains a transaction indicator that is validated against the transaction indicator values. If this is not defined, all operations within a single message will be seen to have occurred within a whole transaction. If defined, then it determines the beginning, middle and end of transactions. Transactions defined in this way can span messages. This is an optional property.

```
fixed.txind={field_name|jms_property}
```

For example:

```
fixed.txind=$TX_IND
```

11.3.2.2.18 fixed.txind.beginval

This value identifies an operation as the beginning of a transaction. The default is B.

```
fixed.txind.beginval={value|\xhex_value}
```

For example:

```
fixed.txind.beginval=0
```

11.3.2.2.19 fixed.txind.middleval

This value identifies an operation as the middle of a transaction. The default is M.

```
fixed.txind.middleval={value|\xhex_value}
```

For example:

```
fixed.txind.middleval=1
```

11.3.2.2.20 fixed.txind.endval

This value identifies an operation as the end of a transaction. The default is E.

```
fixed.txind.endval={value|\xhex_value}
```

For example:

```
fixed.txind.endval=2
```

11.3.2.2.21 fixed.txind.wholeval

This value identifies an operation as a whole transaction. The default is W.

```
fixed.txind.wholeval={value|\xhex_value}
```

For example:

```
fixed.txind.wholeval=3
```

11.3.2.3 Delimited Parser Properties

The following properties are required for the delimited parser except where otherwise noted.

- [delim.sourcedefs](#)

- [delim.header](#)
- [delim.seqid](#)
- [delim.timestamp](#)
- [delim.timestamp.format](#)
- [delim.txid](#)
- [delim.txowner](#)
- [delim.txname](#)
- [delim.optype](#)
- [delim.optype.insertval](#)
- [delim.optype.updateval](#)
- [delim.optype.deleteval](#)
- [delim.schemaandtable](#)
- [delim.schema](#)
- [delim.table](#)
- [delim.txind](#)
- [delim.txind.beginval](#)
- [delim.txind.middleval](#)
- [delim.txind.endval](#)
- [delim.txind.wholeval](#)
- [delim.fielddelim](#)
- [delim.linedelim](#)
- [delim.quote](#)
- [delim.nullindicator](#)
- [delim.fielddelim.escaped](#)
- [delim.linedelim.escaped](#)
- [delim.quote.escaped](#)
- [delim.nullindicator.escaped](#)
- [delim.hasbefores](#)
- [delim.hasnames](#)
- [delim.afterfirst](#)
- [delim.isgrouped](#)
- [delim.dateformat | delim.dateformat.table | delim.dateform.table.column](#)

11.3.2.3.1 delim.sourcedefs

Specifies the location of the source definitions file to use.

`delim.sourcedefs=`*file_location*

For example:

```
delim.sourcedefs=dirdef/hrdemo.def
```

11.3.2.3.2 delim.header

Specifies the list of values that come before the data and assigns names to each.

```
delim.header=name[,name2][. . .]
```

The names must be unique. They can be referenced in other `delim` properties or wherever header fields can be used.

For example:

```
delim.header=optype, tablename, ts  
delim.timestamp=ts
```

11.3.2.3.3 delim.seqid

Specifies the name of the header field, JMS property, or system value that contains the `seqid` used to uniquely identify individual records. This value must increment and the last character must be the least significant.

```
delim.seqid={field_name|jms_property}*seqid}
```

Where:

- *field_name* indicates the name of a header field containing the *seqid*
- *jms_property* uses the value of the specified JMS header property, a special value of this is `$jmsid` which uses the value returned by the mechanism chosen by the `jms.id` property
- *seqid* indicates a simple continually incrementing 64-bit integer generated by the system

For example:

```
delim.seqid=$jmsid
```

11.3.2.3.4 delim.timestamp

Specifies the name of the JMS property, header field, or system value that contains the timestamp.

```
delim.timestamp={field_name|jms_property}*ts}
```

For example:

```
delim.timestamp=TIMESTAMP  
delim.timestamp=$JMSTimeStamp  
delim.timestamp=*ts
```

11.3.2.3.5 delim.timestamp.format

Specifies the format of the timestamp field.

```
delim.timestamp.format=format
```

Where the *format* can include punctuation characters plus:

- YYYY – four digit year
- YY – two digit year

- M[M] – one or two digit month
- D[D] – one or two digit day
- HH – hours in twenty four hour notation
- MI – minutes
- SS – seconds
- Fn – n number of fractions

The default format is "YYYY-MM-DD:HH:MI:SS.FFF"

For example:

```
delim.timestamp.format=YYYY-MM-DD-HH.MI.SS
```

11.3.2.3.6 delim.txid

Specifies the name of the JMS property, header field, or system value that contains the txid used to uniquely identify transactions. This value must increment for each transaction.

```
delim.txid={field_name|$jms_property|*txid}
```

For most cases using the system value of *txid is preferred.

For example:

```
delim.txid=$JMSTxId  
delim.txid=*txid
```

11.3.2.3.7 delim.txowner

Specifies the name of the JMS property, header field, or static value that contains an arbitrary user name associated with a transaction. This value may be used to exclude certain transactions from processing. This is an optional property.

```
delim.txowner={field_name|$jms_property|"value"}
```

For example:

```
delim.txowner=$MessageOwner  
delim.txowner="jsmith"
```

11.3.2.3.8 delim.txname

Specifies the name of the JMS property, header field, or static value that contains an arbitrary name to be associated with a transaction. This is an optional property.

```
delim.txname={field_name|$jms_property|"value"}
```

For example:

```
delim.txname="fixedtx"
```

11.3.2.3.9 delim.optype

Specifies the name of the JMS property or header field that contains the operation type. This is compared to the values for `delim.optype.insertval`, `delim.optype.updateval` and `delim.optype.deleteval` to determine the operation.

```
delim.optype={field_name|$jms_property}
```

For example:

```
delim.optype=optype
```

11.3.2.3.10 delim.optype.insertval

This value identifies an insert operation. The default is I.

```
delim.optype.insertval={value|\xhex_value}
```

For example:

```
delim.optype.insertval=A
```

11.3.2.3.11 delim.optype.updateval

This value identifies an update operation. The default is U.

```
delim.optype.updateval={value|\xhex_value}
```

For example:

```
delim.optype.updateval=M
```

11.3.2.3.12 delim.optype.deleteval

This value identifies a delete operation. The default is D.

```
delim.optype.deleteval={value|\xhex_value}
```

For example:

```
delim.optype.deleteval=R
```

11.3.2.3.13 delim.schemaandtable

Specifies the name of the JMS property or header field that contains the schema and table name in the form SCHEMA.TABLE.

```
delim.schemaandtable={field_name|$jms_property}
```

For example:

```
delim.schemaandtable=$FullTableName
```

11.3.2.3.14 delim.schema

Specifies the name of the JMS property, header field, or hard-coded value that contains the schema name.

```
delim.schema={field_name|$jms_property|"value"}
```

For example:

```
delim.schema="OGG"
```

11.3.2.3.15 delim.table

Specifies the name of the JMS property or header field that contains the table name.

```
delim.table={field_name|$jms_property}
```


For example:

```
delim.table=TABLE_NAME
```

11.3.2.3.16 delim.txind

Specifies the name of the JMS property or header field that contains the transaction indicator to be validated against `beginval`, `middleval`, `endval` or `wholeval`. All operations within a single message will be seen as within one transaction if this property is not set. If it is set it determines the beginning, middle and end of transactions. Transactions defined in this way can span messages. This is an optional property.

```
delim.txind={field_name|$jms_property}
```

For example:

```
delim.txind=txind
```

11.3.2.3.17 delim.txind.beginval

The value that identifies an operation as the beginning of a transaction. The default is B.

```
delim.txind.beginval={value|\xhex_value}
```

For example:

```
delim.txind.beginval=0
```

11.3.2.3.18 delim.txind.middleval

The value that identifies an operation as the middle of a transaction. The default is M.

```
delim.txind.middleval={value|\xhex_value}
```

For example:

```
delim.txind.middleval=1
```

11.3.2.3.19 delim.txind.endval

The value that identifies an operation as the end of a transaction. The default is E.

```
delim.txind.endval={value|\xhex_value}
```

For example:

```
delim.txind.endval=2
```

11.3.2.3.20 delim.txind.wholeval

The value that identifies an operation as a whole transaction. The default is W.

```
delim.txind.wholeval={value|\xhex_value}
```

For example:

```
delim.txind.wholeval=3
```

11.3.2.3.21 delim.fielddelim

Specifies the delimiter value used to separate fields (columns) in the data. This value is defined through characters or hexadecimal values:

```
delim.fielddelim={value|\xhex_value}
```

For example:

```
delim.fielddelim=,  
delim.fielddelim=\xc7
```

11.3.2.3.22 delim.linedelim

Specifies the delimiter value used to separate lines (records) in the data. This value is defined using characters or hexadecimal values.

```
delim.linedelim={value|\xhex_value}
```

For example:

```
delim.linedelim=|  
delim.linedelim=\x0a
```

11.3.2.3.23 delim.quote

Specifies the value used to identify quoted data. This value is defined using characters or hexadecimal values.

```
delim.quote={value|\xhex_value}
```

For example:

```
delim.quote="
```

11.3.2.3.24 delim.nullindicator

Specifies the value used to identify NULL data. This value is defined using characters or hexadecimal values.

```
delim.nullindicator={value|\xhex_value}
```

For example:

```
delim.nullindicator=NULL
```

11.3.2.3.25 delim.fielddelim.escaped

Specifies the value that will replace the field delimiter when the field delimiter occurs in the input field. The syntax is:

```
delim.fielddelim.escaped={value|\xhex_value}
```

For example, given the following property settings:

```
delim.fielddelim=-  
delim.fielddelim.escaped=$#$
```

If the data does not contain the hyphen delimiter within any of the field values:

one two three four

The resulting delimited data is:

one-two-three-four

If there are hyphen (-) delimiters within the field values:

one two three four-fifths two-fifths

The resulting delimited data is:

one-two-three-four\$\$\$fifths-two\$\$\$fifths

11.3.2.3.26 delim.linedelim.escaped

Specifies the value that will replace the line delimiter when the line delimiter occurs in the input data. The syntax is:

```
delim.linedelim.escaped={value|\xhex_value}
```

For example, given the following property settings:

```
delim.linedelim=\  
delim.linedelim.escaped=%/%
```

If the input lines are:

These are the lines and they
do not contain the delimiter.

Because the lines do not contain the backslash (\), the result is:

These are the lines and they\
do not contain the delimiter.\

However, if the input lines do contain the delimiter:

These are the lines\data values
and they do contain the delimiter.

So the results are:

These are the lines%%data values\
and they do contain the delimiter.\

11.3.2.3.27 delim.quote.escaped

Specifies the value that will replace a quote delimiter when the quote delimiter occurs in the input data. The syntax is:

```
delim.quote.escaped={value|\xhex_value}
```

For example, given the following property settings:

```
delim.quote="  
delim.quote.escaped="'"
```

If the input data does not contain the quote (") delimiter:

It was a very original play.

The result is:

"It was a very original play."

However, if the input data does contain the quote delimiter:

It was an "uber-original" play.

The result is:

"It was an ""uber-original"" play."

11.3.2.3.28 `delim.nullindicator.escaped`

Specifies the value that will replace a null indicator when a null indicator occurs in the input data. The syntax is:

```
delim.nullindicator.escaped={value|\xhex_value}
```

For example, given the following property settings:

```
delim.fielddelim=,  
delim.nullindicator=NULL  
delim.nullindicator.escaped=$NULL$
```

When the input data does not contain a NULL value or a NULL indicator:

```
1 2 3 4 5
```

The result is

```
1,2,3,4,5
```

When the input data contains a NULL value:

```
1 2 4 5
```

The result is

```
1,2,NULL,4,5
```

When the input data contains a NULL indicator:

```
1 2 NULL 4 5
```

The result is:

```
1,2,$NULL$,4,5
```

11.3.2.3.29 `delim.hasbefore`

Specifies whether before values are present in the data.

```
delim.hasbefore={true|false}
```

The default is false. The parser expects to find before and after values of columns for all records if `delim.hasbefore` is set to true. The before values are used for updates and deletes, the after values for updates and inserts. The `afterfirst` property specifies whether the before images are before the after images or after them. If `delim.hasbefore` is false, then no before values are expected.

For example:

```
delim.hasbefore=true
```

11.3.2.3.30 delim.hasnames

Specifies whether column names are present in the data.

```
delim.hasnames={true|false}
```

The default is false. If true, the parser expects to find column names for all records. The parser validates the column names against the expected column names. If false, no column names are expected.

For example:

```
delim.hasnames=true
```

11.3.2.3.31 delim.afterfirst

Specifies whether after values are positioned before or after the before values.

```
delim.afterfirst={true|false}
```

The default is false. If true, the parser expects to find the after values before the before values. If false, the after values are before the before values.

For example:

```
delim.afterfirst=true
```

11.3.2.3.32 delim.isgrouped

Specifies whether the column names and before and after images should be expected grouped together for all columns or interleaved for each column.

```
delim.isgrouped={true|false}
```

The default is false. If true, the parser expects find a group of column names (if hasnames is true), followed by a group of before values (if hasbefores), followed by a group of after values (the afterfirst setting will reverse the before and after value order). If false, the parser will expect to find a column name (if hasnames), before value (if hasbefores) and after value for each column.

For example:

```
delim.isgrouped=true
```

11.3.2.3.33 delim.dateformat | delim.dateformat.*table* | delim.dateform.*table*.*column*

Specifies the date format for column data. This is specified at a global level, table level or column level. The format used to parse the date is a subset of the formats used for `parser.timestamp.format`.

```
delim.dateformat=format
```

```
delim.dateformat.TABLE=format
```

```
delim.dateformat.TABLE.COLUMN=format
```

Where:

- *format* is the format defined for `parser.timestamp.format`.
- *table* is the fully qualified name of the table that is currently being processed.
- *column* is a column of the specified table.

For example:

```
delim.dateformat=YYYY-MM-DD HH:MI:SS  
delim.dateformat.MY.TABLE=DD/MM/YY-HH.MI.SS  
delim.dateformat.MY.TABLE.EXP_DATE=YYMM
```

11.3.2.4 XML Parser Properties

The following properties are used by the XML parser.

- [xml.sourcedefs](#)
- [xml.rules](#)
- [rulename.type](#)
- [rulename.match](#)
- [rulename.subrules](#)
- [txrule.timestamp](#)
- [txrule.timestamp.format](#)
- [txrule.seqid](#)
- [txrule.txid](#)
- [txrule.txowner](#)
- [txrule.txname](#)
- [oprule.timestamp](#)
- [oprule.timestamp.format](#)
- [oprule.seqid](#)
- [oprule.txid](#)
- [oprule.txowner](#)
- [oprule.txname](#)
- [oprule.schemandtable](#)
- [oprule.schema](#)
- [oprule.table](#)
- [oprule.optype](#)
- [oprule.optype.insertval](#)
- [oprule.optype.updateval](#)
- [oprule.optype.deleteval](#)
- [oprule.txind](#)
- [oprule.txind.beginval](#)
- [oprule.txind.middleval](#)
- [oprule.txind.endval](#)
- [oprule.txind.wholeval](#)
- [colrule.name](#)
- [colrule.index](#)

- [colrule.value](#)
- [colrule.isnull](#)
- [colrule.ismissing](#)
- [colrule.before.value](#)
- [colrule.before.isnull](#)
- [colrule.before.ismissing](#)
- [colrule.after.value](#)
- [colrule.after.isnull](#)
- [colrule.after.ismissing](#)

11.3.2.4.1 xml.sourcedefs

Specifies the location of the source definitions file.

`xml.sourcedefs=`*file_location*

For example:

`xml.sourcedefs=dirdef/hrdemo.def`

11.3.2.4.2 xml.rules

Specifies the list of XML rules for parsing a message and converting to transactions, operations and columns:

`xml.rules=`*xml_rule_name*[, . . .]

The specified XML rules are processed in the order listed. All rules matching a particular XML document may result in the creation of transactions, operations and columns. The specified XML rules should be transaction or operation type rules.

For example:

`xml.rules=dyntxrule, statoprule`

11.3.2.4.3 rulename.type

Specifies the type of XML rule.

`rulename.type={`*tx|op|col*`}`

Where:

- *tx* indicates a transaction rule
- *op* indicates an operation rule
- *col* indicates a column rule

For example:

`dyntxrule.type=tx`
`statoprule.type=op`

11.3.2.4.4 *rule*.match

Specifies an XPath expression used to determine whether the rule is activated for a particular document or not.

```
rule.match=xpath_expression
```

If the XPath expression returns any nodes from the document, the rule matches and further processing occurs. If it does not return any nodes, the rule is ignored for that document.

The following example activates the *dyntxrule* if the document has a root element of *Transaction*

```
dyntxrule.match=/Transaction
```

Where *statoprule* is a sub rule of *stattxtule*, the following example activates the *statoprule* if the parent rule's matching nodes have child elements of *NewMyTableEntry*.

```
statoprule.match=./NewMyTableEntry
```

11.3.2.4.5 *rule*.subrules

Specifies a list of rule names to check for matches if the parent rule is activated by its match.

```
rule.subrules=xml_rule_name[, . . .]
```

The specified XML rules are processed in the order listed. All matching rules may result in the creation of transactions, operations and columns.

Valid sub-rules are determined by the parent type. Transaction rules can only have operation sub-rules. Operation rules can have operation or column sub-rules. Column rules cannot have sub-rules.

For example:

```
dyntxrule.subrules=dynoprule  
statoprule.subrules=statkeycolrule, statcol1rule
```

11.3.2.4.6 *txrule*.timestamp

Controls the transaction timestamp by instructing the adapter to 1) use the transaction commit timestamp contained in a specified XPath expression or JMS property or 2) use the current system time. This is an optional property.

```
txrule.timestamp={xpath_expression|jms_property|*ts}
```

The timestamp for the transaction may be overridden at the operation level, or may only be present at the operation level. Any XPath expression must end with a value, accessor, such as *@att* or *text()*.

For example:

```
dyntxrule.timestamp=@ts
```

11.3.2.4.7 *txrule*.timestamp.format

Specifies the format of the timestamp field.

```
txrule.timestamp.format=format
```


Where the format can include punctuation characters plus:

- YYYY – four digit year
- YY – two digit year
- M[M] – one or two digit month
- D[D] – one or two digit day
- HH – hours in twenty four hour notation
- MI – minutes
- SS – seconds
- Fn – n number of fractions

The default format is "YYYY-MM-DD:HH:MI:SS.FFF"

For example:

```
dyntxrule.timestamp.format=YYYY-MM-DD-HH.MI.SS
```

11.3.2.4.8 *txrule.seqid*

Specifies the seqid for a particular transaction. This can be used when there are multiple transactions per message. Determines the XPath expression, JMS property, or system value that contains the transactions seqid. Any XPath expression must end with a value accessor such as @att or text().

```
txrule.seqid={xpath_expression|$jms_property|*seqid}
```

For example:

```
dyntxrule.seqid=@seqid
```

11.3.2.4.9 *txrule.txid*

Specifies the XPath expression, JMS property, or system value that contains the txid used to unique identify transactions. This value must increment for each transaction.

```
txrule.txid={xpath_expression|$jms_property|*txid}
```

For most cases using the system value of *txid is preferred.

For example:

```
dyntxrule.txid=$JMSTxId  
dyntxrule.txid=*txid
```

11.3.2.4.10 *txrule.txowner*

Specifies the XPath expression, JMS property, or static value that contains an arbitrary user name associated with a transaction. This value may be used to exclude certain transactions from processing.

```
txrule.txowner={xpath_expression|$jms_property|"value"}
```

For example:

```
dyntxrule.txowner=$MessageOwner  
dyntxrule.txowner="jsmith"
```

11.3.2.4.11 *txrule.txname*

Specifies the XPath expression, JMS property, or static value that contains an arbitrary name to be associated with a transaction. This is an optional property.

```
txrule.txname={xpath_expression|jms_property|"value"}
```

For example:

```
dyntxrule.txname="fixedtx"
```

11.3.2.4.12 *oprule.timestamp*

Controls the operation timestamp by instructing the adapter to 1) use the transaction commit timestamp contained in a specified XPath expression or JMS property or 2) use the current system time. This is an optional property.

```
oprule.timestamp={xpath_expression|jms_property|*ts}
```

The timestamp for the operation will override a timestamp at the transaction level.

Any XPath expression must end with a value accessor such as `@att` or `text()`.

For example:

```
statoprule.timestamp=./CreateTime/text()
```

11.3.2.4.13 *oprule.timestamp.format*

Specifies the format of the timestamp field.

```
oprule.timestamp.format=format
```

Where the *format* can include punctuation characters plus:

- YYYY – four digit year
- YY – two digit year
- M[M] – one or two digit month
- D[D] – one or two digit day
- HH – hours in twenty four hour notation
- MI – minutes
- SS – seconds
- Fn – n number of fractions

The default format is "YYYY-MM-DD:HH:MI:SS.FFF"

For example:

```
statoprule.timestamp.format=YYYY-MM-DD-HH.MI.SS
```

11.3.2.4.14 *oprule.seqid*

Specifies the seqid for a particular operation. Use the XPath expression, JMS property, or system value that contains the operation seqid. This overrides any seqid defined in parent transaction rules. Must be present if there is no parent transaction rule.

Any XPath expression must end with a value accessor such as `@att` or `text()`.

```
oprule.seqid={xpath_expression|$jms_property|*seqid}
```

For example:

```
dynoprule.seqid=@seqid
```

11.3.2.4.15 *oprule.txid*

Specifies the XPath expression, JMS property, or system value that contains the txid used to uniquely identify transactions. This overrides any txid defined in parent transaction rules and is required if there is no parent transaction rule. The value must be incremented for each transaction.

```
oprule.txid={xpath_expression|$jms_property|*txid}
```

For most cases using the system value of `*txid` is preferred.

For example:

```
dynoprule.txid=$JMSTxId  
dynoprule.txid=*txid
```

11.3.2.4.16 *oprule.txowner*

Specifies the XPath expression, JMS property, or static value that contains an arbitrary user name associated with a transaction. This value may be used to exclude certain transactions from processing. This is an optional property.

```
oprule.txowner={xpath_expression|$jms_property|"value"}
```

For example:

```
dynoprule.txowner=$MessageOwner  
dynoprule.txowner="jsmith"
```

11.3.2.4.17 *oprule.txname*

Specifies the XPath expression, JMS property, or static value that contains an arbitrary name to be associated with a transaction. This is an optional property.

```
oprule.txname={xpath_expression|$jms_property|"value"}
```

For example:

```
dynoprule.txname="fixedtx"
```

11.3.2.4.18 *oprule.schemaandtable*

Specifies the XPath expression JMS property or hard-coded value that contains the schema and table name in the form `SCHEMA.TABLE`. Any XPath expression must end with a value accessor such as `@att` or `text()`. The value is verified to ensure the table exists in the source definitions.

```
oprule.schemaandtable={xpath_expression|$jms_property|"value"}
```

For example:

```
statoprule.schemaandtable="MY.TABLE"
```

11.3.2.4.19 *oprule.schema*

Specifies the XPath expression, JMS property or hard-coded value that contains the schema name. Any XPath expression must end with a value accessor such as `@att` or `text()`.

```
oprule.schema={xpath_expression|jms_property|"value"}
```

For example:

```
statoprule.schema=@schema
```

11.3.2.4.20 *oprule.table*

Specifies the XPath expression, JMS property or hard-coded value that contains the table name. Any XPath expression must end with a value accessor such as `@att` or `text()`.

```
oprule.table={xpath_expression|jms_property|"value"}
```

For example:

```
statoprule.table=$TableName
```

11.3.2.4.21 *oprule.optype*

Specifies the XPath expression, JMS property or literal value that contains the optype to be validated against an optype insertval. Any XPath expression must end with a value accessor such as `@att` or `text()`.

```
oprule.optype={xpath_expression|jms_property|"value"}
```

For example:

```
dynoprule.optype=@type  
statoprule.optype="I"
```

11.3.2.4.22 *oprule.optype.insertval*

Specifies the value that identifies an insert operation. The default is I.

```
oprule.optype.insertval={value|\xhex_value}
```

For example:

```
dynoprule.optype.insertval=A
```

11.3.2.4.23 *oprule.optype.updateval*

Specifies the value that identifies an update operation. The default is U.

```
oprule.optype.updateval={value|\xhex_value}
```

For example:

```
dynoprule.optype.updateval=M
```

11.3.2.4.24 *oprule.optype.deleteval*

Specifies the value that identifies a delete operation. The default is D.

```
oprule.optype.deleteval={value|\xhex_value}
```

For example:

```
dynoprule.optype.deleteval=R
```

11.3.2.4.25 *oprule.txind*

Specifies the XPath expression or JMS property that contains the transaction indicator to be validated against `beginval` or other value that identifies the position within the transaction. All operations within a single message are regarded as occurring within a whole transaction if this property is not defined. Specifies the begin, middle and end of transactions. Any XPath expression must end with a value accessor such as `@att` or `text()`. Transactions defined in this way can span messages. This is an optional property.

```
oprule.txind={xpath_expression|$jms_property}
```

For example:

```
dynoprule.txind=@txind
```

11.3.2.4.26 *oprule.txind.beginval*

Specifies the value that identifies an operation as the beginning of a transaction. The default is B.

```
oprule.txind.beginval={value|\xhex_value}
```

For example:

```
dynoprule.txind.beginval=0
```

11.3.2.4.27 *oprule.txind.middleval*

Specifies the value that identifies an operation as the middle of a transaction. The default is M.

```
oprule.txind.middleval={value|\xhex_value}
```

For example:

```
dynoprule.txind.middleval=1
```

11.3.2.4.28 *oprule.txind.endval*

Specifies the value that identifies an operation as the end of a transaction. The default is E.

```
oprule.txind.endval={value|\xhex_value}
```

For example:

```
dynoprule.txind.endval=2
```

11.3.2.4.29 *oprule.txind.wholeval*

Specifies the value that identifies an operation as a whole transaction. The default is W.

```
oprule.txind.wholeval={value|\xhex_value}
```

For example:

```
dynoprule.txind.wholeval=3
```

11.3.2.4.30 *colrule.name*

Specifies the XPath expression or hard-coded value that contains a column name. The column index must be specified if this is not and the column name will be resolved from that. If specified the column name will be verified against the source definitions file. Any XPath expression must end with a value accessor such as `@att` or `text()`.

```
colrule.name={xpath_expression|"value"}
```

For example:

```
dyncolrule.name=@name  
statkeycolrule.name="keycol"
```

11.3.2.4.31 *colrule.index*

Specifies the XPath expression or hard-coded value that contains a column index. If not specified then the column name must be specified and the column index will be resolved from that. If specified the column index will be verified against the source definitions file. Any XPath expression must end with a value accessor such as `@att` or `text()`.

```
colrule.index={xpath_expression|"value"}
```

For example:

```
dyncolrule.index=@index  
statkeycolrule.index=1
```

11.3.2.4.32 *colrule.value*

Specifies the XPath expression or hard-coded value that contains a column value. Any XPath expression must end with a value accessor such as `@att` or `text()`. If the XPath expression fails to return any data because a node or attribute does not exist, the column value will be deemed as null. To differentiate between null and missing values (for updates) the `isnull` and `ismissing` properties should be set. The value returned is used for delete before values, and update/insert after values.

```
colrule.value={xpath_expression|"value"}
```

For example:

```
statkeycolrule.value=../text()
```

11.3.2.4.33 *colrule.isnull*

Specifies the XPath expression used to discover if a column value is null. The XPath expression must end with a value accessor such as `@att` or `text()`. If the XPath expression returns any value, the column value is null. This is an optional property.

```
colrule.isnull=xpath_expression
```

For example:

```
dyncolrule.isnull=@isnull
```

11.3.2.4.34 *colrule.ismissing*

Specifies the XPath expression used to discover if a column value is missing. The XPath expression must end with a value accessor such as `@att` or `text()`. If the XPath expression returns any value, then the column value is missing. This is an optional property.

```
colrule.ismissing=xpath_expression
```

For example:

```
dyncolrule.ismissing=./missing
```

11.3.2.4.35 *colrule.before.value*

Overrides *colrule.value* to specifically say how to obtain before values used for updates or deletes. This has the same format as *colrule.value*. This is an optional property.

For example:

```
dyncolrule.before.value=./beforevalue/text()
```

11.3.2.4.36 *colrule.before.isnull*

Overrides *colrule.isnull* to specifically say how to determine if a before value is null for updates or deletes. This has the same format as *colrule.isnull*. This is an optional property.

For example:

```
dyncolrule.before.isnull=./beforevalue/@isnull
```

11.3.2.4.37 *colrule.before.ismissing*

Overrides *colrule.ismissing* to specifically say how to determine if a before value is missing for updates or deletes. This has the same format as *colrule.ismissing*. This is an optional property.

For example:

```
dyncolrule.before.ismissing=./beforevalue/missing
```

11.3.2.4.38 *colrule.after.value*

Overrides *colrule.value* to specifically say how to obtain after values used for updates or deletes. This has the same format as *colrule.value*. This is an optional property.

For example:

```
dyncolrule.after.value=./aftervalue/text()
```

11.3.2.4.39 *colrule.after.isnull*

Overrides *colrule.isnull* to specifically say how to determine if an after value is null for updates or deletes. This has the same format as *colrule.isnull*. This is an optional property.

For example:

```
dyncolrule.after.isnull=./aftervalue/@isnull
```

11.3.2.4.40 *colrule.after.ismissing*

Overrides *colrule.ismissing* to specifically say how to determine if an after value is missing for updates or deletes. This has the same format as *colrule.ismissing*. This is an optional property.

For example:

```
dyncolrule.after.ismissing=./aftervalue/missing
```


12

Troubleshoot

- [Troubleshooting the Java Adapters](#)

12.1 Troubleshooting the Java Adapters

This chapter includes the following sections:

Topics:

- [Checking for Errors](#)
- [Reporting Issues](#)

12.1.1 Checking for Errors

There are two types of errors that can occur in the operation of Oracle GoldenGate for Java:

- The Replicat process running or VAM does not start or abends
- The process runs successfully, but the data is incorrect or nonexistent

If the Replicat or Extract process does not start or abends, check the error messages in order from the beginning of processing through to the end:

1. Check the Oracle GoldenGate event log for errors, and view the Extract report file:

```
GGSCI> VIEW GGSEVT  
GGSCI> VIEW REPORT {replicat/extract name}
```

2. Check the applicable log file.

For the native log file:

- Look at the last messages reported in the log file for the native library. The file name is the log file prefix (log.logname) set in the property file and the current date.

```
shell> more {log.logname}_{yyyymmdd}.log
```

Note

This is the only log file for the shared library, not the Java application.

3. If the Replicat, or VAM was able to launch the Java runtime, then a log4j log file will exist.

The name of the log file is defined in your log4j.properties file. By default, the log file name is ggjava-version-log4j.log, where version is the version number of the JAR file being used. For example:

```
shell> more ggjava-*log4j.log
```

To set a more detailed level of logging for the Java application, either:

- Edit the current log4j properties file to log at a more verbose level or

- Re-use one of the existing log4j configurations by editing properties file:

```
jvm.bootoptions=-Djava.class.path=ggjava/ggjava.jar  
-Dlog4j.configuration=debug-log4j.properties -Xmx512m
```

These pre-configured log4j property files are found in the classpath, and are installed in:

```
./ggjava/resources/classes/*log4j.properties
```

4. If one of these log files does not reveal the source of the problem, run the native process directly from the shell (outside of GGSCI) so that stderr and stdout can more easily be monitored and environmental variables can be verified. For example:

```
shell> REPLICAT PARAMFILE dirprm/javaue.prm
```

If the process runs successfully, but the data is incorrect or nonexistent, check for errors in any custom filter, formatter or handler you have written.

To restart the Replicat from the beginning of a trail, see [Restarting the Java Delivery](#).

12.1.2 Reporting Issues

If you have a support account for Oracle GoldenGate, submit a support ticket and include the following:

- Operating system and Java versions
The version of the Java Runtime Environment can be displayed by:

```
$ java -version
```
- Configuration files:
 - Parameter file for the Replicat
 - All properties files used, including any JMS or JNDI properties files
 - Velocity templates for formatting purposes
 - If applicable, also include the target-specific configuration file
- Log files:

In the Oracle GoldenGate install directory, all .log files: the Java log4j log files and the native module or VAM log file.