Oracle Cloud Native Environment

Oracle Container Host for Kubernetes Image Builder for Release 2

G17139-08
April 2026

Oracle Cloud Native Environment Oracle Container Host for Kubernetes Image Builder for Release 2,

G17139-08

Contents

Preface

This document includes information on the Oracle Container Host for Kubernetes Image Builder (OCK Image Builder). This is a tool that builds bootable media for Oracle Container Host for Kubernetes (OCK), based on a treefile configuration.

1

Introduction

Describes the Oracle Container Host for Kubernetes Image Builder utility.

The Oracle Container Host for Kubernetes Image Builder (OCK Image Builder) is a tool that builds Oracle Container Host for Kubernetes (OCK) images for Oracle Cloud Native Environment (Oracle CNE) deployments.

For more information about OCK images, see Oracle Cloud Native Environment: Kubernetes Clusters.

Use OCK Image Builder to build custom OCK images for situations where the standard OCK image isn't adequate. For example, consider creating a custom OCK image if you require:

- A different partition layout to the standard OCK image partition layout.
- Extra packages not included in the standard OCK image.
- Device drivers for devices that aren't available in the standard OCK image.

OCK Image Builder consists of a set of shell scripts, the main one of which is `ock-forge`, and image-specific configuration files that can be used and edited as necessary.

For a specific configuration, OCK Image Builder can generate:

- A bootable Qcow2 image, to create cluster nodes.
- An OSTree container image, which Oracle CNE uses to update nodes in a running cluster.

OCK images are configured using rpm-ostree treefiles and therefore require knowledge of OSTree and rpm-ostree.

For more information about OCK Image Builder, see the OCK Image Builder GitHub repository.

2 Installation

Install the OCK Image Builder utility on an Oracle Linux 9 host.

OCK Image Builder must be installed on Oracle Linux 9, from the OCK Image Builder GitHub repository. The prerequisites are Podman, and the `qemu-img` utility.

1. Install Podman.

   ```
   sudo dnf install container-tools
   ```

   For more information on installing and using Podman, see the Oracle Linux: Podman User's Guide.

2. Install `qemu-img`.

   The `qemu-img` utility is used to manipulate Qcow2 images. Install `qemu-ing` from the `ol9_kvm_utils` repository.

   First, enable the repository:

   ```
   sudo dnf config-manager --enable ol9_kvm_utils
   ```

   Then, install `qemu-img`:

   ```
   sudo dnf install qemu-img
   ```

3. Install Git.

   Install the Git client:

   ```
   sudo dnf install git
   ```

4. Install OCK Image Builder.

   Clone the OCK Image Builder GitHub repository:

   ```
   git clone https://github.com/oracle-cne/ock-forge
   ```

5. Install the OCK Configuration files:

   Clone the OCK Configuration GitHub repository. The examples in this guide assume that the configuration files are in the `configs` directory within the OCK Image Builder installation, so perform the clone from within the `ock-forge` directory.

   ```
   cd ock-forge
   git clone https://github.com/oracle-cne/ock
   ```

6. Load the `nbd` kernel module.

   OCK Image Builder can generate a bootable OCK image in Qcow2 format, as a raw disk image, or directly to a physical disk. The `ock-forge` utility mounts Qcow2 images on a

Network Block Device (NBD). Raw disk images are mounted either to a physical disk, or a loopback device, depending on the options specified.

So that OCK Image Builder can mount Qcow2 images to an NBD, load the `nbd` kernel module:

```
sudo modprobe nbd max_part=max_partitions
```

Change *max_partitions* to the maximum number of partitions the resulting Qcow2 image requires. The default is `8`, which is enough for most images.

3
Command Reference

Command reference for the `ock-forge` utility.

The `ock-forge` utility builds bootable media and OSTree container images for OCK, based on an rpm-ostree treefile configuration.

```
ock-forge
[{-s|--source} URI]
[{-b|--branch} branch-name]
[{-d|--device} block-device]
[{-D|--disk} image-path]
[{-f|--filesystem} filesystem]
[{-i|--image} container-image]
[{-s|--source} URI]
[{-b|--branch}]
[{-C|--configs-dir} path]
[{-c|--config-dir} path]
[{-o|--os-name} name]
[{-O|--ostree-image-path} path]
[{-n|--no-clean}]
[{-P|--partition}]
[{-p|--provider} ignition-provider]
[{-I|--ignition} path]
[{-B|--build-image} container-image]
[[--karg key=value]...]
[[--karg-append key=value]...]
[[--karg-delete key=value]...]
```

Where:

{-s|--source} *URI*
The path to either a Git repository or a directory. The contents of the location is copied to the build directory. If this option is omitted, the default configuration is used.

{-b|--branch} *branch-name*
The name of a Git branch. If this option is provided, the indicated branch is cloned. Without this option, the default branch is cloned.

{-d|--device} *block-device*
The path to an existing block device. This device is the installation target. All operations performed by OCK Image Builder are run against this device.

{-D|--disk} *image-path*
The path to a disk image file. This value is only necessary if the installation target is a file rather than a raw device. The disk image file is attached to the device specified with the `--device` option.

{-f|--filesystem} *filesystem*
The file system to use when formatting the root partition. This must be `xfs`.

{-i|--image} ***container-image***
A fully qualified container image name, including a tag. This argument is used in several ways. If provided alone, this container image is used as the base image to deploy the OS. In this case, building the OSTree is skipped entirely. If provided with the `--config-dir` option, and the `--ostree-image-path` option, it generates an OSTree container image that can be used for later installations or upgrades.

{-s|--source} ***URI***
The URI of an OCK configuration. The configuration is copied from this location into the value of the `--configs-dir` argument. If the URI ends with `.git`, the assumption is that it refers to a Git repository. Otherwise, the URI is assumed to be a directory on the local file system.

{-b|--branch}
If `--source` refers to a Git repository, the specified branch is checked out after cloning.

{-C|--configs-dir} ***path***
A directory containing a set of rpm-ostree configurations.

{-c|--config-dir} ***path***
A directory containing the rpm-ostree configuration to build. This must be a subdirectory of `--configs-dir`, at whatever depth is required to ensure all symlinks can be resolved. If this option is specified, a complete rpm-ostree build is performed and optionally packaged into an Open Container Initiative (OCI) archive.

{-o|--os-name} ***name***
The name of the OSTree deployment.

{-O|--ostree-image-path} ***path***
The path to write the Open Container Initiative (OCI) archive generated by the installation process. If this value isn't specified, no archive is generated. If the `--config-dir` option isn't provided, this option is ignored. If the `--image` option is provided and points to a valid OSTree container image, it's used as the reference for generating a chunked image.

{-n|--no-clean}
If this option is provided, don't perform any post install cleanup steps such as unmounting partitions, or detaching virtual block devices.

{-P|--partition}
If this option is provided, the block device specified using the `--device` option has its partition table wiped and repopulated with the default geometry.

{-p|--provider} ***ignition-provider***
Sets the ignition provider. The default value is `qemu`, unless the `--ignition` option is used, in which case the default value is `file`.
This can also be set to `oci` when creating images for OCI. The `oci` provider reads Ignition information from the `metadata` `user_data` key/value pair in the **Instance metadata service**, when creating a compute instance.

{-I|--ignition} ***path***
The path to an ignition file, which `ock-forge` can install directly to the boot disk. This is the complete ignition file from the output of an `ocne cluster start` or `ocne cluster join` command. The file is embedded into the initramfs on the deployment. If the `--provider` option isn't specified, the `--provider` option is set to `file`.

{-B|--build-image} *container-image*
The name of a container image to use when building the OCK boot media. The default value is `ock-builder:latest`.

--karg *key=value*
Set a kernel argument to pass to the `ostree` command. This overrides any existing argument with the same *key*. See the `ostree-admin-deploy(1)` man page for usage. Each kernel argument requires a separate `--karg` option.

--karg-append *key=value*
Append a kernel argument to pass to the `ostree` command. See the `ostree-admin-deploy(1)` man page for usage. Each kernel argument appended requires a separate `--karg-append` option.

--karg-delete *key=value*
Delete an existing kernel argument to the `ostree` command. See the `ostree-admin-deploy(1)` man page for usage. Each kernel argument deleted requires a separate `--karg-delete` option.

4

Build OCK Images

Learn how to use the OCK Image Builder's `ock-forge` utility to build custom images.

This chapter contains information and examples for building OCK images with the `ock-forge` utility.

The ock-forge Script

Describes the capabilities of the `ock-forge` shell script.

The primary component of OCK Image Builder is the `ock-forge` script. The `ock-forge` script generates a bootable OCK image from an rpm-ostree treefile. It requires that the target is a single block device using the standard partition layout. For more information, see Partition Layout.

> **Warning**
>
> Although the `ock-forge` script can build an image from any arbitrary treefile, it's only intended for use with OCK images.

The `ock-forge` script can output generated images in three different formats:

- A Qcow2 image.
- A Raw ISO image.
- To a block device (physical disk).

In addition to generating the bootable image, `ock-forge` creates an OSTree container image. This container image is used by Oracle CNE to update nodes in a running cluster.

Configuration Files

Describes OCK image configuration files.

The `ock-forge` script requires an rpm-ostree treefile configuration that specifies what the generated image contains. Because the script generates OCK images that are intended for Oracle CNE deployments, the treefile must contain the base configuration that's required to work in that environment. These configuration files are available for different target Kubernetes versions in the OCK Configuration GitHub repository.

We recommend that you don't edit these configuration files directly, but include any custom configuration in separate files. This prevents custom configuration from being overwritten by future updates to the OCK image configuration files.

The initial configuration file that the `ock-forge` script reads is called `manifest.yaml`. Use an existing configuration for the target Kubernetes version and customize it by including extra configuration files in the `manifest.yaml` file as follows:

```
# manifest.yaml
include:
- base.yaml
- ux.yaml
- ocne.yaml
- removals.yaml
- custom/myconfig.yaml
...
```

For example, to make the Vim editor available in the custom OCK image, add the following to the `myconfig.yaml` file:

```
packages:
- vim
```

For more information on the available parameters in an rpm-ostree treefile, see the rpm-ostree Treefile reference.

Partition Layout

Describes the standard partition layout required by OCK images.

Installing to a disk with existing partitions requires at least the following partitions:

- An EFI partition.
- A boot partition labelled `boot`.
- A root partition labelled `root`.

We recommend the disk contains only these three partitions and that the root partition is the last partition on the disk. The root partition must be last so that the partition can be automatically expanded to fill the entire disk when the OS boots for the first time. Other layouts are possible, but these have limited applicability and require a good understanding of how partitions are laid out and used.

Building OCK Images Examples

Provides examples of using the `ock-forge` script to build and customize standard OCK images.

This chapter shows how to use the Oracle Container Host for Kubernetes Image Builder to build standard OCK images for Oracle CNE deployments, and how these OCK images can be customized using first-boot host customization through `extraIgnitionInline` using Butane.

- Building Standard OCK Images
- Customizing Standard OCK Images

Building Standard OCK Images

Provides examples of using the `ock-forge` script to build standard OCK images.

Example 4-1 Building a typical Qcow2 image

A typical invocation builds Qcow2 images. The `ock-forge` script does all the work required. This example generates a new Qcow2 image, attaches it as a block device, partitions the disk, formats the partitions, installs the OS, and generates an OSTree archive:

```
sudo ./ock-forge -d /dev/nbd0 -D out/1.33/boot.qcow2 \
                 -i container-registry.oracle.com/olcne/ock-ostree:1.33 \
                 -O ./out/1.33/archive.tar \
                 -C ./ock -c configs/config-1.33 -P
```

Example 4-2 Building a Qcow2 image from GitHub

The `ock-forge` script can copy configurations from inconvenient places to more convenient places. This example builds a Qcow2 and OSTree image from scratch, using the OCK GitHub repository as a source of truth. The clone of the repository is retained and can be reused in later invocations:

```
sudo ./ock-forge -d /dev/nbd0 -D out/1.33/boot.qcow2 \
                 -i container-registry.oracle.com/olcne/ock-ostree:1.33 \
                 -O ./out/1.33/archive.tar \
                 -C ./ock -c configs/config-1.33 \
                 -s https://github.com/oracle-cne/ock.git -P
```

Example 4-3 Build a raw disk image

This example generates a raw disk image, rather than a Qcow2 image. The generated image can be `dd`'ed onto a physical disk, and used to boot a system directly:

```
sudo ./ock-forge -d /dev/loop0 -D out/1.33/boot.iso \
                 -i container-registry.oracle.com/olcne/ock-ostree:1.33 \
                 -O ./out/1.33/archive.tar \
                 -C ./ock -c configs/config-1.33 -P
```

Example 4-4 Install to a physical disk

This example installs the image to a physical block device, creating the necessary partitions:

```
sudo ./ock-forge -d /dev/sdb \
                 -i container-registry.oracle.com/olcne/ock-ostree:1.33 \
                 -O ./out/1.33/archive.tar \
                 -C ./ock -c configs/config-1.33 -P
```

Example 4-5 Install but don't generate OSTree archive

This example performs a fresh installation of the OS, but doesn't store the contents in an OSTree container image archive:

```
sudo ./ock-forge -d /dev/nbd0 -C ./ock -c configs/config-1.33 -P
```

Example 4-6 Install from a container image

This example installs the OS using an existing OSTree container image as the source:

```
sudo ./ock-forge -d /dev/nbd0 -d /dev/loop0 -D out/1.33/boot.iso \
                -i container-registry.oracle.com/olcne/ock-ostree:1.33 -P
```

Customizing Standard OCK Images

Provides examples of using Ignition files to customize standard OCK images.

Customizing standard OCK images

The following examples show how to use a YAML configuration file, located in the `~/.ocne/defaults.yaml` directory, which conforms to the Butane schema to generate an Ignition config file that's used to customize a standard OCK image through `extraIgnitionInline` during its first boot.

Example 4-7 Create a user

This example creates a new user:

```
extraIgnitionInline: |
  variant: fcos
  version: 1.5.0

  passwd:
    users:
      - name: NAME_OF_USER
        home_dir: PATH_TO_YOUR_HOME_DIRECTORY
        groups:
          - NAME_OF_USERS_GROUP
        ssh_authorized_keys:
          - YOUR_PUBLIC_KEY
```

Example 4-8 Create a directory

This example shows how to create a directory and assign permissions to it:

```
extraIgnitionInline: |
  variant: fcos
  version: 1.5.0

  storage:
    directories:
      - path: PATH_TO_DIRECTORY_TO_CREATE
        mode: OCTAL_FILE_PERMISISON_VALUE
        user:
          name: USERS_NAME
        group:
          name: USERS_GROUP
```

Example 4-9 Create a text file in a directory

This example shows how to create a file in a directory and assign permissions to it:

```
extraIgnitionInline: |
  variant: fcos
  version: 1.5.0

  storage:
    files:
      - path: PATH_TO_FILE_TO_CREATE
        mode: OCTAL_FILE_PERMISISON_VALUE
        user:
          name: USERS_NAME
        group:
          name: USERS_GROUP
        overwrite: true
        contents:
          inline: |
            Hello from Oracle CNE on OCK.
            This file was created at first boot by extraIgnitionInline.
```

Example 4-10 Change the Message of the Day file

This example shows how to change the Message of the Day file:

```
extraIgnitionInline: |
  variant: fcos
  version: 1.5.0

  storage:
    files:
      - path: /etc/motd
        mode: 0644
        overwrite: true
        contents:
          inline: |
            Oracle CNE custom OCK node
            Example user customization enabled
            Current Kubernetes version: 1.33
```

Example 4-11 Set the root partition size

This example shows how to set the root partition size to 30Gb and use the rest of the disk as another partition:

```
extraIgnitionInline: |
  variant: fcos
  version: 1.5.0

  storage:
    disks:
    - device: /dev/sda
      wipe_table: false
      partitions:
      - label: lvm1
        number: 4
        start_mib: 30720
```

Example 4-12 Enable a first-boot systemd Service

This example shows how to configure `systemd` to enable a first-boot audit service:

```
extraIgnitionInline: |
  variant: fcos
  version: 1.5.0
```

```
  systemd:
    units:
      - name: bootstrap-audit.service
        enabled: true
        contents: |
          [Unit]
          Description=Audit first boot customization
          After=network-online.target
          Wants=network-online.target

          [Service]
          Type=oneshot
          ExecStart=/usr/bin/bash -c 'date > /var/log/first-boot-audit.log'

          [Install]
          WantedBy=multi-user.target
```

Example 4-13 Create and merge two disk partitions then create a new mount point.

This example shows how to create two partitions, then join them using Logical Volume Manager and mount them to a new mount point (`/var/lvtest`):

```
extraIgnitionInline: |
  variant: fcos
  version: 1.5.0

  storage:
    disks:
    - device: /dev/sda
      wipe_table: false
      partitions:
      - label: lvm1
        number: 4
        start_mib: 30720
        size_mib: 70000
      - label: lvm2
        number: 5
    directories:
      - path: /var/lvtest
    files:
      - path: /etc/lvscript.sh
        mode: 755
        contents:
          inline: |
            #! /bin/bash
            set -x
            if [ -b /dev/mapper/bdgroup-bdvol ]; then exit 0; fi
            pvcreate /dev/sda4 /dev/sda5
            vgcreate bdgroup /dev/sda4 /dev/sda5
            lvcreate -L 30GB -n bdvol bdgroup
            mkfs.xfs /dev/mapper/bdgroup-bdvol
            UUID=$(blkid -s UUID -o value /dev/mapper/bdgroup-bdvol)
            systemctl set-environment BDVOL_UUID=$UUID
            mkdir /etc/systemd/system/var-lvtest.mount.d
            cat > /etc/systemd/system/var-lvtest.mount.d/uuid.conf << EOF
            [Mount]
            Environment=BDVOL_UUID=$UUID
            EOF
  systemd:
    units:
      - name: lvsetup.service
        enabled: true
```

```
      contents: |
        [Service]
        Type=oneshot
        ExecStart=/etc/lvscript.sh

        [Install]
        WantedBy=multi-user.target
    - name: var-lvtest.mount
      enabled: true
      contents: |
        [Unit]
        After=lvsetup.service

        [Mount]
        What=UUID=${BDVOL_UUID}
        Where=/var/lvtest
        Type=xfs
        Options=defaults

        [Install]
        WantedBy=multi-user.target
```

Example 4-14 Define several services in a single file

This example shows how to showing how several services could be defined in a single definition file:

```
extraIgnitionInline: |
  variant: fcos
  version: 1.5.0

  passwd:
    users:
      - name: appuser
        # Sets the intended home directory for the account.
        home_dir: /home/appuser
        # Adds the user to the wheel group for administrative access.
        groups:
          - wheel
        # Replace this with a real public key for customer testing.
        ssh_authorized_keys:
          - ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI_REPLACE_WITH_YOUR_PUBLIC_KEY

  storage:
    directories:
      - path: /home/appuser
        # Explicitly creates the user's home directory and sets ownership.
        mode: 0755
        user:
          name: appuser
        group:
          name: appuser

    files:
      - path: /home/appuser/README.txt
        # Creates a text file in the user's home directory.
        mode: 0644
        user:
          name: appuser
        group:
          name: appuser
        overwrite: true
        contents:
```

```
      inline: |
        Hello from Oracle CNE on OCK for Kubernetes 1.32.
        This file was created at first boot by extraIgnitionInline.

  - path: /etc/motd
    mode: 0644
    overwrite: true
    contents:
      inline: |
        Oracle CNE custom OCK node
        Example user customization enabled
        Kubernetes target: 1.32
```

5
OCK Builder Utilities

Lists the individual scripts that the OCK Image Builder tool consists of.

The `ock-forge` utility is a shell script that relies on other shell scripts to do its work. Each of the shell scripts can be useful by itself, but probably requires editing to make it so.

> **Warning**
>
> Ensure that you understand how these scripts function before changing them. These utility scripts can be destructive if used incorrectly.

The shell scripts that make up the OCK Image Builder tool are listed in the following table.

Table 5-1 OCK Image Builder Utility Scripts

Script	Options	Notes
`setup-vm-disk.sh`	`-d` *`network-block-device`* `-D` *`path-to-new-image`*	Creates a Qcow2 image and attaches it to a network block device.
`sparsify-image.sh`	`-D` *`path-to-image`*	Re-sparsifies and compresses a Qcow2 image. It does part of the job of `virt-sparsify` and the result isn't as good. However, it has the advantage of not requiring a VM and can be used in environments where starting a VM isn't an option. For example, building an image inside an ARM VM. It requires several gigabytes of storage while running. That space is released by the time the script ends.
`make-partitions.sh`	`-d` *`path-to-device`* `-f` *`filesystem`*	Partitions a disk using a standard layout. It creates a small EFI partition as partition 1, a small boot partition as partition 2, and a root partition with the rest of the disk as the last partition.

Example 5-1 Creating and attaching Qcow2 images

To create a Qcow2 image and attach it to `/dev/nbd0`:

```
sudo ./setup-vm-disk -d /dev/nbd0 -D mydisk.qcow2
```

Example 5-2 Sparsify a Qcow2 image

To sparsify and compress an existing Qcow2 image:

```
sudo ./sparsify-image.sh -D mydisk.qcow2
```

Example 5-3 Partition a disk

To partition a physical block device using XFS:

```
sudo ./make-partitions.sh -d /dev/sdb -f xfs
```