Oracle Linux 9

KVM User's Guide

G25090-07
March 2026

Oracle Linux 9 KVM User's Guide,

G25090-07

Contents

5 KVM Instances: Hardware Configuration

6 KVM Known Issues

Preface

Oracle Linux 9: KVM User's Guide provides information about how to install, configure, and use the Oracle Linux KVM packages to run guest system on top of a bare metal Oracle Linux system. This documentation provides information on using KVM on a standalone platform in an unmanaged environment. Typical usage in this mode is for development and testing purposes, although production level deployments are supported. Oracle recommends that customers use Oracle Linux Virtualization Manager for more complex deployments of a managed KVM infrastructure.

1

Deployment Overview: Oracle Linux KVM

For a high-level overview of the Kernel-based Virtual Machine (KVM) deployment options, operating requirements, and virtualization package descriptions, see these topics:

- KVM Management: Deployment Options
- KVM Guest: Operating Systems
- KVM Host: System Requirements
- KVM Virtualization Packages
- Virtualization Package Sources

KVM Management: Deployment Options

Oracle Linux offers two KVM deployment management options: Standalone KVM Hypervisor and Managed KVM Server Virtualization.

Standalone KVM Hypervisor

This KVM option provides a set of modules that enable you to use the Oracle Linux kernel as a hypervisor. KVM can be used on both x86_64 and aarch64 processor architectures and is available on Oracle Linux 9 systems using either Red Hat Compatible Kernel (RHCK) or Unbreakable Enterprise Kernel (UEK).

KVM features are actively developed and might vary depending on platform and kernel release. Information on how to use the KVM hypervisor option is described in this guide.

Oracle UEK users have the option of using either the default kernel version or an Oracle kernel version for KVM deployment. UEK users can also switch between the KVM stacks as needed.

For more details about the virtualization packages available for each Linux release, see Virtualization Package Sources.

For details about switching between KVM stacks, see Switch KVM Stacks.

> **Note**
>
> For UEK users, consult the kernel version release notes for more information about KVM features and any known issues that might apply. For more information, see the Unbreakable Enterprise Kernel documentation.

> **Note**
>
> Instead of using the CLI to manage KVM instances on a host, you can use the Cockpit web console. This provides a graphical interface to interact with KVM and `libvirt`. For more details, see Oracle Linux: Using the Cockpit Web Console.

Managed KVM Server Virtualization

This KVM option is for enterprise or clustered KVM deployment environments on Oracle Linux. For these KVM environments, consider using Oracle Linux Virtualization Manager which is a server virtualization management platform.

> **Note**
>
> Oracle Linux Virtualization Manager is available for Oracle Linux 8 only.

Using Oracle Linux Virtualization Manager's administration or virtual machine (VM) portals, you can configure, monitor, and manage an Oracle Linux KVM environment, including hosts, VMs, storage, networks, and users. Oracle Linux Virtualization Manager also provides a REST API for managing Oracle Linux KVM infrastructure, enabling you to integrate Oracle Linux Virtualization Manager with other management systems or to automate repetitive tasks with scripts.

For details on how to use Oracle Linux Virtualization Manager, see the Oracle Linux Virtualization Manager documentation.

KVM Guest: Operating Systems

When installing standalone instances of KVM, consider using one of the following guest OS:

- Linux Guest OS
- Microsoft Windows Guest OS
- Oracle Solaris Guest OS

Linux Guest OS

An Oracle Linux host system can run the following Linux OS as KVM guests.

> **Note**
>
> Only 64-bit Linux OS are supported for KVM guests on Oracle Linux host systems.

- Oracle Linux 7
- Oracle Linux 8
- Oracle Linux 9
- Oracle Linux 10
- Oracle Container Host for Kubernetes
- Red Hat Enterprise Linux 7
- Red Hat Enterprise Linux 8
- Red Hat Enterprise Linux 9
- Red Hat Enterprise Linux 10

- Centos 7
- Centos 8
- Centos Stream 8
- Centos Stream 9
- Centos Stream 10
- AlmaLinux OS 8
- AlmaLinux OS 9
- AlmaLinux OS 10
- Rocky Linux 8
- Rocky Linux 9
- Rocky Linux 10
- Suse Linux Enterprise Server 12
- Suse Linux Enterprise Server 15
- Ubuntu 16.04
- Ubuntu 18.04
- Ubuntu 20.04
- Ubuntu 22.04
- Ubuntu 24.04

> **Note**
>
> Oracle Linux ISO images and disk images are available for download from Oracle Software Delivery Cloud: https://edelivery.oracle.com/linux.

Microsoft Windows Guest OS

The following Microsoft Windows versions support the use of guest OS installations on KVM instances:

- Microsoft Windows Server 2025
- Microsoft Windows Server 2022
- Microsoft Windows Server 2019
- Microsoft Windows Server 2016
- Microsoft Windows 11
- Microsoft Windows 10 (64-bit only)

VirtIO Driver Requirements

For improved performance with network and block (disk) devices, and to resolve common issues, we recommend that you install the Oracle VirtIO Drivers for Microsoft Windows. These drivers are para-virtualized drivers for Microsoft Windows guests running on Oracle Linux KVM hypervisors.

Microsoft Windows guests on KVM have been tested by using the Oracle VirtIO Drivers for Microsoft Windows. For instructions on how to obtain and install the drivers, see Oracle Linux: Oracle VirtIO Drivers for Microsoft Windows for use with KVM.

Oracle Solaris Guest OS

Oracle Solaris version 11.4 supports the use of guest OS installations on KVM instances.

Oracle Solaris 11.4 can be used as a guest OS when installed within a standalone instance of KVM.

> **Note**
>
> Oracle Solaris version 11.4.33 (Oracle Solaris 11.4 SRU 33) is the minimum version that includes functionality for VirtIO guest support.

Oracle Solaris ISO images and disk images are available for download from the Oracle Software Delivery Cloud at https://edelivery.oracle.com/.

> **Tip**
>
> For best results when using Oracle Solaris as a guest OS:
>
> - Use at least a two-core configuration for the Oracle Solaris VM.
> - Use the most current QEMU system type (Custom Emulated Machine = pc-i440fx-4.2) for the Oracle Solaris VM.

KVM Host: System Requirements

Many of the system requirements for KVM hosts can depend on the kinds of applications running on the virtual machine (VM) and the amount of work they're expected to perform.

The following table describes the minimum system requirements and suggested guidelines for deploying KVM hosts.

Bare metal host	KVM can be used when it's run on a bare metal host. Note that nested virtualization scenarios aren't supported for KVM deployments.
CPU	The host system CPU must have virtualization features for Intel (VT-x) or AMD (AMD-V) enabled. Arm (aarch64) CPUs can also be used. If virtualization features aren't available, check that virtualization is enabled in the system firmware BIOS or UEFI. As a rule of thumb, you can start with the following virtual CPU to host CPU ratios (this ratio is of distinct CPU cores and assumes SMT is enabled):

	• 1:1 to 2:1 can typically achieve good VM performance. • 3:1 may cause some VM performance degradations. • 4:1 or greater might cause significant VM performance problems. The ratio of virtual CPUs to host CPUs can be calculated by running performance tests on VM and host systems. Deciding on acceptable performance depends on many factors such as, for example: • Tasks that VM systems perform. • Volume of tasks to be processed. • Preferred rate that these tasks need to be processed.
Memory	3 GB reserved for the host is a good starting point but memory requirements for the host OS scale with the amount of physical memory available. For systems with lots of available physical memory, increase the reserved memory for the host OS. For example, on a system with 1 TB memory, We recommend at least 20 GB available for the host OS. If system work on a host and all VMs starts exceeding the available physical RAM, the performance impact is severe. However, if VMs are typically idle, you might not need to allocate as much RAM. Ensure you do performance testing to ensure that applications always have enough memory.
Storage	The minimum disk space required for the host OS is typically 6 GB. Each VM requires its own storage for the guest OS and for swap usage. Cater to around 6 GB, at minimum, per VM that you intend to create, but consider the purpose of the VM and scale accordingly.

KVM Virtualization Packages

To run Oracle Linux KVM on a host, you must install the Oracle Linux KVM virtualization packages. Several package groups are available to match different needs and use cases.

For installation instructions, see KVM Host: Installation and Configuration.

Recommended Virtualization Packages

The following individual packages are recommended for installation on virtualization hosts.

Package	Description
`libvirt`	Provides an interface to KVM, and the `libvirt` daemons for managing guest VMs. The Cockpit web console also provides a graphical interface that interacts with KVM and `libvirt` to set up and configure VMs on a system. See Oracle Linux: Using the Cockpit Web Console for more information.
`qemu-kvm`	Installs the QEMU emulator that performs hardware virtualization so that guests can access host CPU and other resources.
`qemu-img`	Provides functionality that lets you create, convert, and change images offline. It works with all QEMU image formats. Never use `qemu-img` to change images in use by a running virtual machine or any other process. Doing so might destroy the image.
`virt-install`	Provides command line utilities for creating and provisioning guest VMs.
`virt-viewer`	Provides a graphical utility that can be loaded into a desktop environment to access the graphical console of a guest VM. For information about connecting to a KVM graphical interface using `virt-viewer`, see the `virt-viewer(1)` manual page.

Recommended Virtualization Package Groups

Virtualization packages can also be installed from package groups. Group packages contain the minimum set of packages that are required for a virtualization host.

Package Group	Description
`"Virtualization Hypervisor"`	Contains the minimum set of packages that are required for a virtualization host.
`"Virtualization Tools"`	Contains tools for managing virtual images offline.
`"Virtualization Host"`	Not available for the aarch64 platform. Contains `"Virtualization Hypervisor"` and `"Virtualization Tools"`
`"Virtualization Client"`	Available for installation on GUI environment Oracle Linux systems.

Downloading Virtualization Packages

Virtualization packages are available for download from the Oracle Linux yum server or from the Unbreakable Linux Network (ULN). The virtualization packages are also available from various upstream projects, including:

- https://www.linux-kvm.org/page/Main_Page
- https://libvirt.org/
- https://www.qemu.org/

Virtualization Package Sources

Oracle distributes two sets of KVM user space packages, called *stacks*. These KVM stacks define which versions of virtualization hypervisor packages install and run on an Oracle Linux system.

Each stack has its own separate source, so you might need to configure the package sources on the system depending on which KVM stack you want to install.

> **Note**
>
> If the system is running UEK, you can switch between stacks. For more details, see Switch KVM Stacks.

The following table lists the KVM stacks available for deployment to an Oracle Linux 9 system, their benefits, and support limitations, and their corresponding package sources.

Table 1-1 Oracle Linux: KVM Stacks and Package Sources

KVM Stack	Benefits and Limitations	Yum Repository	Installation Notes
Default KVM Stack	• Fully supported across all Oracle Linux kernels. • Offers maximum compatibility with RHCK and Red Hat Enterprise Linux.	`ol9_appstream` (Enabled by default on all Oracle Linux 9 systems)	The Default KVM stack can be installed without changing any repository configuration.
Oracle KVM Stack	• Engineered to work with KVM features that are enabled in the latest releases of UEK. • Provides QEMU 10.1 and libvirt 12.0. • Requires UEK8U2 (UEK Release 8 Update 2).	`ol9_kvm_utils`	Before enabling this repository, you must remove all existing virtualization packages. If you remain on UEK7 (or do not enable UEK8 repositories), you stay on the QEMU 7.2 and libvirt 9.0 version of the Oracle KVM Stack.

For more information about enabling repositories using `dnf`, or for details about how Oracle manages software package distribution using yum repositories and ULN channels, see:

- Oracle Linux: Managing Software on Oracle Linux
- Oracle Linux: Unbreakable Linux Network User's Guide

2

KVM Host: Installation and Configuration

Install user-space packages, enable `libvirt` services, and validate that an Oracle Linux 9 system can host KVM virtual machines.

Use this procedure to prepare an Oracle Linux 9 system to host KVM virtual machines. It installs the required packages, enables the `libvirt` services, and verifies that the host supports hardware virtualization. Complete this task on any host before creating or managing VMs.

1. Optional: If you require the Oracle KVM stack instead of the Default KVM stack, then Enable Yum Repositories.

   > **Important**
   >
   > This step is only required when using the Oracle KVM Stack. The Default KVM Stack is available in the standard core system repositories.

   To identify the sources that include the Default KVM Stack or the Oracle KVM Stack, see Virtualization Package Sources.
2. Install Virtualization Packages.
3. Validate Host System to ensure that the system can host guest VMs.
4. Optional: On UEK systems only, Switch KVM Stacks between the Default KVM stack and Oracle KVM stack.

Enable Yum Repositories

To use any of the Oracle KVM stacks, first enable the `ol9_kvm_utils` yum repository before installing the virtualization packages.

Ensure that the following are true before proceeding:

- The system meets the virtualization package requirements . See KVM Host: System Requirements and Virtualization Package Sources.
- UEK8U2 (UEK Release 8 Update 2) is installed and running on the system.
- You have administrator privileges.

This task is only required when using Oracle KVM stacks. The Default KVM stack is available in the standard yum repositories and can be installed without enabling any other repositories.

> **Tip**
>
> If the Oracle KVM stack is already installed and you want to switch KVM stacks, see Switch KVM Stacks.

> **Important**
>
> The Oracle KVM stack packages in `ol9_kvm_utils` provide QEMU 10.1 and libvirt 12.0 and have explicit dependencies on UEK8U2 kernel packages. If the system is running UEK7 and UEK8 repositories are not enabled/available, you remain on the QEMU 7.2 and libvirt 9.0.

Follow these steps to enable the `ol9_kvm_utils` repository.

1. Install or update the release package:

   ```
   sudo dnf install -y oraclelinux-release-el9
   ```

2. Enable the Oracle Linux 9 yum repository, by running the `dnf config-manager` command:

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

After enabling the `ol9_kvm_utils` repository, you can install the Oracle KVM Stack. See Install Virtualization Packages.

Install Virtualization Packages

The following information describes how to install the virtualization packages on an Oracle Linux 9 system.

Ensure that the following are true before proceeding:

- You have administrator privileges.
- If installing the Oracle KVM stack:
  - The `ol9_kvm_utils` yum repository must be enabled. See Enable Yum Repositories for instructions.
  - The host must be running the latest UEK R7 or UEK8U2.

Follow these steps to install the virtualization packages on an Oracle Linux 9 system.

1. Sign in to the Oracle Linux 9 system.
2. Ensure that the latest packages are installed on the system:

   ```
   sudo dnf update
   ```

3. Install the base virtualization packages and other utilities:

   - x86_64 Systems
   - aarch64 Systems

x86_64 Systems

– To install the Default KVM stack:

```
sudo dnf group install "Virtualization Host"
sudo dnf install qemu-kvm virt-install virt-viewer
```

– To install the Oracle KVM stack, first remove any existing virtualization packages and then install the packages as usual:

```
sudo dnf remove libvirt qemu-kvm edk2-ovmf
sudo dnf group install "Virtualization Host"
sudo dnf install qemu-kvm virt-install virt-viewer
```

> **Important**
>
> The Oracle KVM stack provides QEMU 10.1 and libvirt 12.0 and requires UEK8U2. If UEK8 repositories are enabled and required packages are available, installing or upgrading the Oracle KVM stack also upgrades the host kernel to UEK8U2. If you remain on UEK7 (or don't enable UEK8 repositories), you stay on the Default KVM stack (QEMU 7.2 and libvirt 9.0).

aarch64 Systems

– To install the Default KVM stack:

```
sudo dnf group install "Virtualization Hypervisor" "Virtualization
Tools"
sudo dnf install qemu-kvm virt-install virt-viewer
```

– To install the Oracle KVM stack, first remove any existing virtualization packages and then install the packages as usual:

```
sudo dnf remove libvirt qemu-kvm edk2-aarch64
sudo dnf group install "Virtualization Hypervisor" "Virtualization
Tools"
sudo dnf install qemu-kvm virt-install virt-viewer
```

> **Important**
>
> The Oracle KVM stack provides QEMU 10.1 and libvirt 12.0 and requires UEK8U2. If UEK8 repositories are enabled and required packages are available, installing or upgrading the Oracle KVM stack also upgrades the host kernel to UEK8U2. If you remain on UEK7 (or do not enable UEK8 repositories), you stay on the Default KVM stack (QEMU 7.2 and libvirt 9.0).

4. Reboot the system to ensure that the virtualization packages and utilities were updated.
5. Ensure that the `libvirt` daemons are running.

Before you can create and manage KVM instances, the `libvirtd` daemons must be started and enabled. See Manage the Libvirt Daemons for instructions.

6. Verify that the system can act as act as a virtual host. See Validate Host System
7. After verifying that the system can act as a virtual host, you can proceed to KVM Instances: Create and Manage.

Validate Host System

Check that the system can host VMs.

The `libvirt` package provides a validation utility that checks whether a system can function correctly as a virtualization host. The utility can check for several virtualization capabilities, but KVM functionality is covered by testing the `qemu` virtualization type.

1. Run the `virt-host-validate qemu` command to validate the host system:

   ```
   sudo virt-host-validate qemu
   ```

2. Evaluate the output.

   If all checks return a `PASS` value, the system can host guest VMs. If any of the tests fail, a reason is provided and information is displayed on how to resolve the issue, if available.

> **Warning**
>
> If the following message is displayed, the system isn't capable of functioning as a KVM host:
>
> ```
> QEMU: Checking for hardware virtualization: FAIL (Only emulated
> CPUs are
>       available, performance will be significantly limited)
> ```
>
> If you try to create or start a VM on a host where this message is displayed, the action is likely to fail.

Switch KVM Stacks

You can switch from one KVM stack to another KVM stack, if required.

The KVM stacks define the versions of user space packages that you can use to manage virtual machines on Oracle Linux. To take advantage of newer functionality available in more recent user space packages, you must install UEK.

The Default KVM stack is compatible with both RHCK and UEK. The Default KVM stack is available from the standard Oracle Linux package repositories. If you require RHCK, you can only use the Default KVM stack.

The Oracle KVM stack, which includes more KVM features than the default stack, requires UEK to be running on the host system and that the `ol9_kvm_utils` yum repository is enabled on the system.

Switching the KVM stack on an Oracle Linux system involves removing one version of the KVM stack and then replacing it with another version of the KVM stack.

> **Caution**
>
> Guest instances that were created by using one KVM stack might not be compatible and might not start after switching to another KVM stack.

For instructions on how to switch KVM stacks on Oracle Linux 9 systems with UEK, see these topics:

- Switch Default Stack to Oracle KVM Stack
- Switch Oracle KVM Stack to Default Stack

> **Note**
>
> For details about the KVM stacks and their sources, see Virtualization Package Sources.

Switch Default Stack to Oracle KVM Stack

The following information describes how Oracle Linux 9 users can switch from the Default KVM stack to the Oracle KVM stack.

Ensure that the following are true before proceeding:

- The Oracle Linux 9 system has the Default KVM stack packages installed.
- The host system must be running the latest UEK7 or UEK8U2.
- You have administrator privileges.

> **Caution**
>
> Guest instances that were created using the default KVM stack might not be compatible and might not start after switching to the Oracle KVM stack.

Follow these steps to switch from the Default KVM stack to the Oracle KVM Stack.

1. Remove any packages from the existing Default KVM Stack, for example:

   ```
   sudo dnf remove libvirt* qemu* virt-install
   ```

2. Enable the `ol9_kvm_utils` yum repository.

   See Enable Yum Repositories.

3. Install the required virtualization packages from the `ol9_kvm_utils` yum repository:

   ```
   sudo dnf install libvirt qemu-kvm virt-install
   ```

> **Important**
>
> The Oracle KVM stack provides QEMU 10.1 and libvirt 12.0 and requires UEK8U2. When you install or upgrade these packages, `dnf` upgrades the host kernel to UEK8U2 as required, if UEK8 repositories are enabled and required packages are available.

Switch Oracle KVM Stack to Default Stack

The following information describes how Oracle Linux 9 users can switch from the Oracle KVM stack to the Default KVM stack.

Ensure that the following are true before proceeding:

- The Oracle Linux 9 system has the Oracle KVM stack packages installed.
- The host system is running the latest UEK7 or UEK8U2.
- You have administrator privileges.

> **Caution**
>
> Guest instances that were created using the Oracle KVM stack might not be compatible and might not start after switching to the Default KVM stack.

Follow these steps to switch from the Oracle KVM stack to the Default KVM stack.

1. Remove any packages from the existing Oracle KVM stack, for example:

   ```
   sudo dnf remove libvirt* qemu* virt-install
   ```

2. Disable the `ol9_kvm_utils` yum repository:

   ```
   sudo dnf config-manager --disable ol9_kvm_utils
   ```

3. Install the required virtualization packages from the `ol9_appstream` yum repository:

   ```
   sudo dnf install libvirt qemu-kvm virt-install
   ```

3
Manage the Libvirt Daemons

The following information describes how to start, enable, and check the status of the `libvirt` daemons on an Oracle Linux 9 KVM host.

Ensure that you have the following before proceeding:

- Virtualization packages installed on the host system. See Install Virtualization Packages for details.
- An understanding of the `libvirt` driver daemons as of Oracle Linux 9. For details, see Types of `libvirt` Driver Daemons
- Administrator privileges.

Follow these steps to start and enable the `libvirt` daemons:

1. To start the `libvirt` daemons with full virtualization functionality, run the following command:

   ```
   for drv in qemu network nodedev nwfilter secret storage interface;
     do
      sudo systemctl enable virt${drv}d.service;
      sudo systemctl enable virt${drv}d{,-ro,-admin}.socket;
      sudo systemctl start virt${drv}d{,-ro,-admin}.socket;
     done
   ```

   You don't need to start the service for each daemon, as the service is automatically started when the first socket is established.

   > **Note**
   >
   > For legacy systems, you can use the `libvirtd` socket to manage remote virtual guest connections.

2. Optional: Enable the `virtproxyd` daemon to let remote hosts connect to guests.

   If connections from remote hosts are needed, the `virtproxyd` daemon must be enabled and started:

   ```
   sudo systemctl enable virtproxyd.service
   sudo systemctl enable virtproxyd-tls.socket
   sudo systemctl start virtproxyd-tls.socket
   ```

3. To check the status of the `libvirt` daemons, run the following command:

   ```
   sudo systemctl list-units --type=socket virt*
   ```

   The output shows all enabled units and their current status.

Types of libvirt Driver Daemons

Oracle Linux9 provides functionality for two different types of libvirt driver daemons: Modular and Monolithic. The granularity in which you can configure individual virtualization drivers depends on which libvirt daemon you use. For example:

- **Modular libvirt** - Oracle Linux 9 Fresh Install
  Modular libvirt, which is newly introduced in Oracle Linux 9, provides a specific daemon for each hypervisor driver. These include:
  - virtqemud: is the QEMU management daemon, for running virtual machines on KVM.
  - virtnetworkd: is the virtual network management daemon.
  - virtnodedevd: is the host physical device management daemon.
  - virtnwfilterd: is the host firewall management daemon.
  - virtsecretd: is the host secret management daemon.
  - virtstoraged: is the host storage management daemon.
  - virtinterfaced: is the host Network Interface Card (NIC) management daemon.
  - virtproxyd is a virtualization proxy daemon that lets remote clients to securely access the libvirt APIs.

  The name of the daemon reflects the name of the host driver, for example: virt[*DRIVER*]d. Each driver daemon has a separate configuration file that resides in the libvirt directory. For example, the configuration file path for the QEMU management driver daemon is /etc/libvirt/virtqemud.conf.

  Modular driver daemons provide better options for fine-tuning and managing the libvirt system resources. When you perform a fresh install of Oracle Linux 9, the libvirt modular virtualization driver daemons are configured by default.

  > **Note**
  >
  > When the virt$[*DRIVER*]d daemon is managed by systemd other features are also available, most notably socket activation. For more information about the use of modular sockets and systemd integration, see https://libvirt.org/daemons.html#modular-sockets.

- **Monolithic libvirt** - Update to Oracle Linux 9

  By default, the traditional monolithic daemon, known as libvirtd is configured when you update from Oracle Linux 8 to Oracle Linux 9. The libvirtd daemon controls a wide variety of virtualization drivers by using a single configuration file (/etc/libvirt/libvirtd.conf). In some instances, system resources might be used inefficiently when using the libvirtd centralized configuration. Therefore, we recommend that Oracle Linux 9 users switch to the modular libvirt driver daemons. For instructions, see https://libvirt.org/daemons.html#switching-to-modular-daemons.

For more information about the libvirt daemons, see https://libvirt.org/daemons.html.

4

KVM Instances: Create and Manage

To create and manage KVM instances from the CLI, see the following topics.

- KVM Instances: Hardware Configuration
- Add Watchdog Device to KVM Instance
- Add vTPM Security to KVM Instance
- KVM Memory and CPU Allocation Configuration
- KVM Storage Configuration
- Overview: Virtual Networking

> **Note**
>
> To create and manage KVM instances using a graphical user interface (GUI), see the *Virtual Machines Management Tasks* topics in the Oracle Linux: Using the Cockpit Web Console. Additionally, you can choose to manage a virtual machine environment by using the Oracle VM Manager. For more details about using the VM Manager, see Oracle Linux Virtualization Manager documentation.

Create: KVM Instance

Create a virtual machine using the `virt-install` utility.

Ensure that:

- The KVM host has been prepared, as described in KVM Host: Installation and Configuration.
- At least one virtual storage pool exists. This is required to create a virtual machine. Note that a virtualization storage pool is automatically provided in the `/var/lib/libvirt/images` directory. See Storage Pools: Create and Manage for more details.
- The virtual machine has a compatible guest OS. See KVM Guest: Operating Systems for more details.
- You have administrator privileges.

Follow these steps to create a virtual machine using the CLI.

1. To create a virtual machine, use the `virt-install` command and its options to define the resources required.

   The syntax is as follows:

   ```
   virt-install \
   --name [unique-virtual-machine-name] \
   --memory [allocated-MiB-memory-size] \
   --vcpus [integer-vcpus-value-for-guest] \
   --location [installation-source-path] \
   ```

```
--disk [type-and-size] \
--os-variant [os-verison] \
--network [default] \
--graphics [none]
```

> **Note**
>
> To list the `virt-install` options, type: `virt-install --help`. Further information on each option is available in the `virt-install(1)` manual page

Where:

`--name` [unique-virtual-machine-name]	(Required) Provide a unique name to the virtual machine. The assigned name is registered as a domain within `libvirt`.
`--memory` [allocated-MiB-memory-size]	(Required) Specify the amount of memory to be allocated to the guest, in MiB.
`--vcpus [integer-vcpus-value-for-guest]`	(Required) Specify the number of virtual CPUs available for use by guest OS.

`--disk [type-and-size]`	(Required) Identify the appropriate values for disk type and size, for example:
	• Disk hardware size:
	`--disk size=[capacity size in gigabytes]`
	• Disk storage pool path
	`--disk /storage-pool-path/volume-path,size=#`
	• Disk image media file path
	`--disk=/iso-images/ol8-dvd.iso,device=cdrom`
	Special considerations:
	• If a `path` isn't specified the disk image is automatically created as a qcow file format.
	• If `virt-install` is run as root, the disk image is created in `/var/lib/libvirt/images/` and is named using the name specified for the VM at install.
	• If `virt-install` is run as an ordinary user, the disk image is created

	in `$HOME/.local/share/libvirt/images/`
`--location [installation-source-path]`	(Required) Specify the OS installation source location, for example: • ISO file • An expanded installation resource hosted at a local path • An expanded installation resource hosted remotely on an HTTP or NFS server. Other source options are listed in the `virt-install (1)` Linux manual page.
`--os-variant [os-version]`	(Optional) Use this option to optimize the performance of the guest configuration. To obtain valid `--os-variant` values, type: `osinfo-query os`
`--network [default]`	(Optional) When the `--network` option isn't specified, or when the `--network default` option is specified, the guest connects to the default network. For other network configuration options, see the `virt-install (1)` Linux manual page.
`--graphics [none]`	(Optional) Specify the display type used for interactive guest installation. When `--graphics none` is specified, a text-only installation display is available. To display a graphical console for a guest installation, you can use the `virt-viewer` tool. For more information about configuring the virt-viewer, see the `virt-viewer(1)` Linux manual page.

Based on the `virt-install` options specified, the virtual machine is created and the guest OS is automatically installed.

Virtual Machine Creation Examples: See Virt-Install: Command Line Examples

2. After creating the virtual machine, you can:
   a. Start the virtual machine.

      For details, see KVM: Start Instance.
   b. Connect to the virtual machine.

Virt-Install: Command Line Examples

The following information provides command-line examples for using `virt-install` to create virtual machine instances with a guest OS.

Guest OS: Oracle Linux 9

- Scenario: ISO image - text-mode only install.

```
virt-install \
    --name ol9-guest-demo --memory 16384 --vcpus 16 --disk size=280 \
    --os-variant ol9.0 --location ol9.iso \
    --graphics none --extra-args='console=ttyS0'
```

  Description: Creates a KVM instance named *ol9-guest-demo* using an `ol9.iso` image file in text-only mode, without graphics. It connects the guest console to the serial console. The VM has 16384 MiB of memory, 16 vCPUs, and 280 GiB disk. Note that this guest installation example might be useful when connecting to a host over a slow network link.

- Scenario: URL installation tree path - automated Kickstart install.

```
virt-install \
    --graphics vnc \
    --name ol9-guest-demo1 --memory 2048 --vcpus 2 --disk size=160 \
    --os-variant ol9.0 --location http://example.com/OS-install \
    --initrd-inject /home/uniquename/ks.cfg --extra-args="inst.ks=file:/
ks.cfg console=tty0 console=ttyS0,115200n8"
```

  Description: Creates a VM named *ol9-guest-demo1* that installs from the `http://example.com/OS-install` URL. Note for the installation to start successfully, the URL must contain a working OS installation tree. The OS is automatically configured by the referenced kickstart file (`/home/uniquename/ks.cfg`). Finally, the VM is allocated with 2048 MiB of RAM, 2 vCPUs, and a 160 GiB qcow2 virtual disk.

  Additions for ARM 64 host-based scenarios:

```
%packages
-kernel
kernel-64k
%end
```

  These lines ensure that the kickstart file, depicted in the *ol9-guest-demo1* scenario, installs the `kernel-64k` package.

Guest OS: Oracle Linux 8

- Scenario: ISO image - live CD

```
virt-install \
    --name ol8-guest-demo --memory 4096 --vcpus 4 \
    --disk none --livecd --os-variant ol8.0 \
    --cdrom /home/uniquename/Downloads/ol8.iso
```

Description: Creates a VM named *ol8-guest-demo* by using `/home/uniquename/Downloads/ol8.iso` image to run a Oracle Linux 8 OS from a live CD. No disk space is assigned to this VM, so any changes made during the session aren't preserved. The VM is allocated with 4096 MiB of RAM and 4 vCPUs.

- Scenario: Import disk image - qcow file format

```
virt-install \
    --name ol8-guest-demo --memory 2048 --vcpus 2 \
    --os-variant ol8.0 --import \
    --disk /home/uniquename/backup/disk.qcow2
```

Description: Creates a VM named *ol8-guest-demo* that connects to an existing disk image (`/home/uniquename/backup/disk.qcow2`). The VM is allocated with 2048 MiB of RAM and 2 vCPUs. Note that the `os-variant` option is highly recommended when importing a disk image. In cases when the `os-variant` option isn't specified, the performance of the created VM might be negatively affected.

Clone: Existing KVM Instance

System administrators can easily create a KVM instance by cloning the configuration of an existing KVM instance. Depending on the use of the clone, system administrators can either prepare the source KVM configuration before cloning it, or simply create a KVM clone with the identical configuration as the source KVM instance.

> **Note**
>
> When creating multiple clones from a single KVM instance, we recommend preparing the source configuration before cloning it. Preparing the source configuration lets you examine the configuration and remove unique parameters that would not apply to the clone configuration and cause the clone to possibly fail or not work correctly.

For instructions on how to prepare a KVM instance for cloning or create a KVM clone, see these topics:

- Prepare KVM for Cloning: Using `virt-sysprep`
- Prepare KVM for Cloning: Manually
- Create a KVM Clone Using `virt-clone` Command

> **Note**
>
> In addition to using the CLI to create KVM clones, you can use the Cockpit web console to clone KVM instances. For details, see *Cloning VMs* in Oracle Linux: Using the Cockpit Web Console.

Prepare KVM for Cloning: Using `virt-sysprep`

The following information describes how to use the system preparation scripting tool (`virt-sysprep`) to prepare a source KVM disk configuration for cloning.

> **Note**
>
> The `virt-sysprep` tool helps you to prepare a KVM configuration for cloning by removing SSH host keys, persistent network configurations, and user accounts on the disk image. It also lets you add SSH keys, users, or logos. For more details about `virt-sysprep`, see https://libguestfs.org/virt-sysprep.1.html.

What Do You Need?

- All important data on the source KVM is backed up.
  Note that `virt-sysprep` changes the disk image in place without making a copy of it. To keep the configuration of the source KVM intact, create a clone. For details, see Create a KVM Clone Using `virt-clone` Command.
- The system preparation tool (`virt-sysprep`) must installed on the host. The tool is included in the `guestfs-tools` package.

  ```
  sudo dnf install guestfs-tools
  ```

- The source KVM must be shut down.
- The location of the source KVM disk image is required. Also, you must be the disk image owner and have disk write permissions.

Follow these steps to use the `virt-sysprep` tool to prepare a source KVM disk image configuration for cloning.

1. Log in as the root owner of the KVM disk image, for example:

   ```
   whoami
   root
   ```

2. To prepare the source KVM disk image for cloning, use the following *virt-sysprep* command syntax.

   ```
   virt-sysprep -a [/var/lib/libvirt/images/a-replace me -my-kvm.qcow2]
   [   0.0] Examining the guest ...
   [   7.3] Performing "abrt-data" ...
   [   7.3] Performing "backup-files" ...
   [   9.6] Performing "bash-history" ...
   [   9.6] Performing "blkid-tab" ...
   [...]
   ```

   Where:

   - [*/var/lib/libvirt/images/replace me-my-kvm.qcow2*] replace with the source KVM disk image path.

3. Use the `virt-sysprep` tool to inspect the prepared disk image. For a list of available options, type: `virt-sysprep --help`

   For more details, see the `virt-sysprep(1)` Linux manual page.

4. After preparing the disk image for cloning, proceed with using the prepared KVM disk configuration to create a clone.

   For details, see Create a KVM Clone Using `virt-clone` Command.

> **Note**
>
> On the first boot of the clone, we recommend that you change the hostname.

Prepare KVM for Cloning: Manually

The following information describes how to manually prepare a source KVM configuration for cloning.

What Do You Need?

- All important data on the source KVM is backed up.
  To keep the configuration of the source KVM intact, create a clone. For details, see Create a KVM Clone Using `virt-clone` Command
- The location of the source KVM disk image is required. Also, you must be the disk image owner and have disk write permissions.
- The source KVM must be shut down.
- Administrator privileges.

Follow these steps to manually prepare a source KVM configuration for cloning.

1. Configure the source KVM as required, for example:
   - Install any required software for clone.
   - Configure any required properties that are considered non-unique for the operating system or system applications.
2. Remove the network configuration, as follows:
   a. To remove any persistent `udev` rules, type:

      ```
      rm -f /etc/udev/rules.d/70-persistent-net.rules
      ```

      > **Note**
      >
      > If you don't remove the udev rules, the name of the first NIC might be `eth1`instead of `eth0`.

   b. (Guests running Oracle Linux 9 or later) Remove any hardware addresses from NetworkManager connection profiles.

      Check for connection profiles in the following locations:

      - `/etc/NetworkManager/system-connections`
      - `/run/NetworkManager/system-connections`
      - `/usr/lib/NetworkManager/system-connections`

   c. (Guests running Oracle Linux 8 or earlier) Change `/etc/sysconfig/network-scripts/ifcfg-eth[x]` to remove the `HWADDR` and static lines and any other unique or non-desired settings, such as UUID.

      For example:

      ```
      DEVICE=eth[x]
      BOOTPROTO=none
      ```

```
ONBOOT=yes
#NETWORK=10.0.1.0       <- REMOVE
#NETMASK=255.255.255.0  <- REMOVE
#IPADDR=10.0.1.20       <- REMOVE
#HWADDR=xx:xx:xx:xx:xx  <- REMOVE
#USERCTL=no             <- REMOVE
```

After modification, the file must not include a `HWADDR` entry or any unique information, and at a minimum include the following lines:

```
DEVICE=eth[x]
ONBOOT=yes
```

> **Important**
>
> You must remove the `HWADDR` entry because if its address doesn't match the new guest's MAC address, the `ifcfg` is ignored.

d. (Guests running Oracle Linux 8 or earlier) If the following files exist, ensure they have the same content:

- `/etc/sysconfig/networking/profiles/default/ifcfg-eth[x]`
- `/etc/sysconfig/networking/devices/ifcfg-eth[x]`

> **Note**
>
> If `NetworkManager` was used or any special settings with the KVM, ensure that all unique information is removed from the `ifcfg` scripts.

3. Remove ULN registration details.

   For example, if the KVM guest from which you want to create a clone is registered with ULN, you must unregister it. For more information, see the applicable reference:

   - Oracle Linux 7 guests: Oracle Linux: Unbreakable Linux Network User's Guide for Oracle Linux 6 and Oracle Linux 7
   - Oracle Linux 8, Oracle Linux 9, or Oracle Linux 10 guests: Oracle Linux: Managing Software on Oracle Linux

4. Remove `sshd` public/private key pairs.

   For example, type:

   ```
   rm -rf /etc/ssh/ssh_host_[name]
   ```

5. Remove any other application-specific identifiers or configurations that might cause conflicts if running on multiple machines.

6. Configure the relevant setup configuration wizard to run at first boot.

   Examples:

- For Oracle Linux 7, enable the first boot and initial-setup wizard:

```
sed -ie 's/RUN_FIRSTBOOT=NO/RUN_FIRSTBOOT=YES/' /etc/sysconfig/
firstboot
systemctl enable firstboot-graphical
systemctl enable initial-setup-graphical
```

- For Oracle Linux 8, Oracle Linux 9, or Oracle Linux 10, remove the `gnome-initial-setup-done` file to configure the KVM to run the configuration wizard on the next boot:

```
# rm ~/.config/gnome-initial-setup-done
```

7. After addressing all required modifications, proceed to using the prepared KVM configuration to create a clone.

   For details, see Create a KVM Clone Using `virt-clone` Command.

> **Note**
>
> On the first boot of the clone we recommend that you change the hostname.

Create a KVM Clone Using `virt-clone` Command

The following information describes how to clone a source KVM instance using the `virt-clone` command.

What Do You Need?

- Root privileges.
- Source KVM is shut down.
- Sufficent disk space to store the cloned disk images.
- (Optional) Prepared source KVM configuration for cloning.
  See Prepare KVM for Cloning: Using `virt-sysprep` or Create a KVM Clone Using `virt-clone` Command.

Steps

Follow these steps to clone an existing KVM instance:

1. Perform one of the following:
   - To clone a source KVM with its original configuration, use the following `virt-clone` command syntax:

     ```
     virt-clone --orginal kvm-name --auto-clone
     ```

     For example, if you typed:

     ```
     virt-clone --orginal My_KVM --auto-clone
     ```

     Output similar to the following appears:

     ```
     virt-clone --original My_KVM --auto-clone
     Allocating 'My_KVM-clone.qcow2'                              | 55.0 GB
     ```

```
00:02:37

Clone 'My_KVM-clone' created successfully.
```

In this example, the `virt-clone` command copies the source My-KVM configuration and creates: (1) a clone KVM guest named `My_KVM-clone`, and (2) a clone disk image named `My_KVM-clone.qcow2`

-OR-

- To clone a source KVM using the `virt-clone` command with other options, type the following command to view the available configuration options:

```
virt-clone --help
```

For more examples of how to use the `virt-clone` command, see the `virt-clone(1)` Linux man page.

2. Verify that the cloned KVM instance is working, for example:
   - Confirm that the clone has been added to the list of KVMs on the host:

```
virsh list --all
Id   Name                  State
---------------------------------------
-    My_KVM          shut off
-    My_KVM-clone    shut off
```

   - Start the clone and observe if it boots up:

```
virsh start My_KVM-clone
Domain 'My_KVM-clone' started
```

View: KVM Instances, Status, and Configuration

The following information describes how to use the `virsh` command to obtain a list of KVM instances available on a host, along with their status (running, paused, and so on). It also describes how to view and edit the XML configuration of a specific KVM instance.

What Do You Need?

- Root privileges on host system.
- Existing KVM instance on a host system.

Steps

Follow theses steps to: 1) view KVM instances and their status on a host, and 2) view basic and detailed configuration information about a specific KVM instance.

1. To list all virtual machines on a local host system, type:

```
virsh list --all
```

Example output:

```
 Id    Name                           State
----------------------------------------------------
 1     My_KVM_Guest                   running
```

Where:

- KVM ID or Name: The unique ID and name assigned to the KVM instance.
- State: The operating status of the KVM instance. Possible status states that might appear include:
  - Running – The KVM instance is considered operational and working.
  - Paused – Execution of the KVM instance has been paused until it's resumed.
  - Shut off or Shutdown – The KVM instance is powered off.
  - Saved – The saved state is similar to the paused state, however the KVM instance's configuration is saved to persistent storage.

  For more details on how to manage the state of a KVM instance, see Start, Shutdown, Reboot, or Remove KVM.

2. To view basic details about a specific KVM instance, type:

   ```
   sudo virsh dominfo My_KVM_Guest
   ```

   Example output:

   ```
   Id:             1
   Name:           My_KVM_Guest
   UUID:           c321630AA-2f5e-665c-8949-75b7d99999e1
   OS Type:        hvm
   State:          running
   CPU(s):         2
   CPU time:       188.3s
   Max memory:     4188304 KiB
   Used memory:    4188304 KiB
   Persistent:     yes
   Autostart:      disable
   Managed save:   no
   Security model: selinux
   Security DOI:   0
   Security label: system_u:system_r:svirt_t:s0:###,### (enforcing)
   ```

3. To view the complete XML configuration associated with a KVM instance, use the `virsh dumpxml` command. For example:

   ```
   sudo virsh dumpxml My_KVM_Guest
   ```

   The `virsh dumpxml` command output returns the guest KVM XML configuration file, which can be viewed, saved, changed, or used in other ways.

4. To edit the XML configuration file associated with a KVM guest, use the `virsh edit` command. For example:

   ```
   sudo virsh edit My_KVM_Guest_Name
   ```

Connect to KVM: `virsh` Serial Console

You can access a KVM instance directly using a serial console interface.
The following information describes how to establish a serial console connection to a KVM instance by using the `virsh console` command.

> **Note**
>
> A serial console connection to a KVM instance might be useful when a the instance isn't configured with a GUI display, or if the instance lacks a network configuration, preventing SSH access.

What Do you Need?

- Administrator privileges
- Name of the KVM instance.
- The KVM instance is configured for serial console use. For example:
  - KVM serial console device defined – Ensure that a serial console device is defined on KVM. For example:

    ```
    sudo virsh ttyconsole kvm_name
    ```

    If output is shown, a serial console device is defined. Otherwise, define a serial console in the KVM XML configuration. One method you can use is `virsh edit`. For example, run:

    ```
    sudo virsh edit
    ```

    Within the `<devices>` tag, add the following text to the XML. For example:

    ```
    <devices>
      <console type='pty'/>
    </devices>
    ```

    See Domain XML format for more information.
  - KVM kernel console option enabled – Ensure that `console=ttyS0` kernel option is enabled on KVM. If this option isn't configured, the `virsh console` connection to the serial console will be unresponsive.
    To verify the `console=ttyS0` kernel option is configured on the KVM, use the `cat /proc/cmdline` command. The output configuration must include `console=[console-name]`. If the output doesn't include a console configuration, you must enable the `console=ttyS0` kernel option.

To enable the `console=ttyS0` kernel option, (1) type:

```
sudo grubby --update-kernel=ALL --args="console=ttyS0"
```

(2) Ensure that the changes were applied, type:

```
sudo grub2-editenv - unset kernelopts
```

(3) Reboot the KVM instance.

Steps

Follow these steps to directly connect to the KVM serial console.

- On the host system, use the `virsh console` command to open up a KVM session in a serial console. For example:

  ```
  sudo virsh console testguest
  ```

  Where `testguest` is the name of the KVM guest.
  You can interact with the `virsh` serial console in the same way as you would with the CLI.

> **Note**
>
> If the connection failed and the guest serial console is unresponsive, you can exit the connection by pressing: **`Ctrl`** key and the **`]`** right square bracket key.

Start, Shutdown, Reboot, or Remove KVM

When using the CLI, you can use the following methods to start, shutdown, reboot, or remove a KVM instance:

- KVM: Start Instance
- KVM: Shut Down Instance
- KVM: Suspend or Resume Instance
- KVM: Reboot Instance
- KVM: Remove KVM Instance

KVM: Start Instance

The following information describes how to start a KVM instance that's shut down on a local or remote host using the `virsh start` command.

What Do You Need?

- Administrator privileges.
- Name of the inactive KVM instance.
- For remote KVM instances, the following is required to complete the remote example shown in Step 1.
  - The host IP address where the inactive KVM instance resides.

- Root privileges to the host.
- SSH connection protocol port enabled.
- The `qemu-kvm` virtualization package is installed. For details about virtualization packages, see KVM Virtualization Packages.

Steps

Follow these steps to start an inactive KVM on a host system using the `virsh start` command:

- Perform one of the following:
  - For local KVM, use the `virsh start` command as follows:

    ```
    sudo virsh start KVM_Guest_Name
    ```

    Example output:

    ```
    Domain 'KVM_guest_name' started
    ```

  - For remote KVM, use the `virsh start` command and the SSH connection protocol as follows:

    ```
    sudo virsh -c qemu+ssh://root@host_ip_address/system start
    Remote_KVM_guest_name
    ```

    Example output:

    ```
    root@host_ip-address's password:

    Domain 'remote_KVM_guest_name' started
    ```

KVM: Shut Down Instance

The following information describes how to shut down an active KVM instance on a local or remote host using the `virsh shutdown` command. It also describes how to force an unresponsive KVM instance on a host to shut down using the `virsh destroy` command.

> **Note**
>
> The `virsh destroy` command doesn't delete or remove the KVM configuration or its disk images. It only forces the running KVM instance to shut down, similarly to pulling the power cord on a physical machine. However, in unique cases, the `virsh destroy` command might cause corruption to the KVM file system, so using this command is only recommended when all other shutdown methods have failed.

What Do You Need?

- Administrator privileges.
- Name of the active or unresponsive KVM instance.

- For remote KVM instances, the following is required to complete the remote example shown in Step 1.
  - The host IP address where the KVM instance resides.
  - Root privileges to the host.
  - SSH connection protocol port enabled.
  - The `qemu-kvm` virtualization package is installed. For details about virtualization packages, see KVM Virtualization Packages.

Steps

Follow these steps to shut down a KVM instance on a host system using either the `virt shutdown` command or the `virt destroy` command.

- Perform one of the following:
  - **Graceful KVM Shutdown** – Perform either of the following:
    - For a local KVM, use the `virt shutdown` command as follows:

      ```
      sudo virsh shutdown KVM_Guest_Name
      ```

      Example output:

      ```
      Domain 'KVM_guest_name' is being shutdown
      ```

    - For a remote KVM, use the `virt shutdown` command and the SSH connection protocol as follows:

      ```
      sudo virsh -c qemu+ssh://root@host_ip_address/system shutdown
      Remote_KVM_guest_name
      ```

      Example output:

      ```
      root@host_ip-address's password:

      Domain 'remote_KVM_guest_name'is being shutdown
      ```

  - **Forceful KVM Shutdown** – Use the `virt destroy` command on an unresponsive KVM instance as follows:

    ```
    sudo virsh destroy KVM_Guest_Name
    ```

    Example output:

    ```
    Domain 'KVM_guest_name' destroyed
    ```

KVM: Suspend or Resume Instance

The following information describes how to suspend an active KVM instance on a local or remote host using the `virsh suspend` command. It also describes how to resume a suspended KVM instance on a host using the `virsh resume` command.

What Do You Need?

- Administrator privileges.
- Name of the active or suspended KVM instance.
- For remote KVM instances, the following is required to complete the remote example shown in Step 1.
  - The host IP address where the KVM instance resides.
  - Root privileges to the host.
  - SSH connection protocol port enabled.
  - The `qemu-kvm` virtualization package is installed. For details about virtualization packages, see KVM Virtualization Packages.

Steps

Follow these steps to suspend or resume a KVM instance on a host system.

- Perform one of the following:
  - **Suspend KVM** – Perform either of the following:
    - For a local KVM, use the `virsh suspend` command as follows:

      ```
      sudo virsh suspend KVM_Guest_Name
      ```

      Example output:

      ```
      Domain 'KVM_guest_name' suspended
      ```

    - For a remote KVM, use the `virsh suspend` command and the SSH connection protocol as follows:

      ```
      sudo virsh -c qemu+ssh://root@host_ip_address/system suspend
      Remote_KVM_guest_name
      ```

      Example output:

      ```
      root@host_ip-address's password:

      Domain 'remote_KVM_guest_name' suspended
      ```

  - **Resume KVM** – Use the `virsh resume` command on a suspended KVM instance as follows:

    ```
    sudo virsh resume KVM_Guest_Name
    ```

    Example output:

    ```
    Domain 'KVM_guest_name' resumed
    ```

KVM: Reboot Instance

The following information describes how to reboot a KVM instance on a local host using the `virsh reboot` command.

> **Note**
>
> Rebooting a KVM can be helpful with various problems and might even be necessary to complete some configurations.

What Do You Need?

- Administrator privileges.
- Name of the KVM instance.

Steps

Follow these steps to reboot a KVM instance on a host system.

- Type:

  ```
  sudo virsh reboot My_KVM_Guest[--mode method]
  ```

  Where:

  - *[--mode method]* is optional. The mode method option lets you specify an alternative shutdown method such as acpi or agent.

  Example output:

  ```
  Domain My-KVM_Guest is being rebooted
  ```

KVM: Remove KVM Instance

The following information describes how to remove a KVM instance on a host system using the `virsh undefine` command. It also describes how to optionally remove the storage artifacts associated with a KVM instance.

What Do You Need?

- Administrator privileges.
- KVM information and actions required:
  - KVM instance name.
  - KVM storage is not in use by other KVMs.
  - Removal of any KVM snapshots. To remove snapshots associated with KVM instance, use the `virsh snapshot-delete`.
  - KVM storage file path. To identify the storage file path associated with a KVM instance, use the `virsh dumpxml` command. For example:

    ```
    sudo virsh dumpxml --domain My_KVMGuest_Name | grep 'source file'
    ```

    Example output:

    ```
    <source file='/home/testuser/.local/share/libvirt/images/
    My_KVMGuest_Name-1.qcow2'/>
    ```

  - KVM instance is shutdown. For details, see KVM: Shut Down Instance

– (Optional) Back up all important data on KVM instance. If required, see Clone: Existing KVM Instance.

Steps

Follow these steps to remove a KVM instance from a host system.

1. To delete the KVM instance, type:

   ```
   sudo virsh undefine My_KVMGuest_Name
   ```

   The `virsh undefine` command removes all configuration information about the KVM instance from libvirt. Note that the associated KVM storage artifacts such as virtual disks remain intact.

2. (Optional) To delete the storage artifacts such as virtual disks associated with the KVM instance (removed in Step 1), use the `rm` command followed by the storage path.

   For example:

   ```
   sudo rm /home/testuser/.local/share/libvirt/images/
   My_KVMGuest_Name-1.qcow2
   ```

Migrate a KVM Guest Using virsh

Use the `virsh` utility to move a KVM guest to another host.

Ensure that the following are true before proceeding:

- The KVM guest is shut down on the source host.

  > **Note**
  >
  > For live migration options, see Managed KVM Server Virtualization.

- You have administrator privileges on the source and destination hosts.
- The source and destination hosts must be mutually reachable over the network, and all required migration ports must be open on the destination host.
- CPUs on the source and destination hosts must be compatible. If the KVM guest configuration requests a CPU model or features not available on the destination, the migrated guest might fail to start.
- Shared storage (for example, NFS/iSCSI/Ceph) must be visible to both hosts, or use `--copy-storage-all` to transfer disks during migration.
- The `libvirt` daemons must be enabled on both hosts. See Manage the Libvirt Daemons .

Follow these steps to migrate a KVM guest from one host to another.

1. Confirm that the KVM guest is shut down on the source host:

   ```
   sudo virsh list --all
   ```

If the `State` field in the output doesn't show the KVM guest as `shut off`, shut it down before proceeding:

```
sudo virsh shutdown KVM_Guest_Name
```

2. (Optional) If using shared storage, ensure that the KVM guest's disks are on a shared path visible to both hosts (for example, an NFS pool).

   See Storage Pools: Create and Manage.

3. Migrate the KVM guest configuration to the destination host.

   Run the following command to re-create the KVM guest configuration on the destination host so that it survives service or host reboots:

   ```
   sudo virsh migrate --offline --persistent KVM_Guest_Name \
       qemu+ssh://destination-host/system
   ```

   If you're not using shared storage, add `--copy-storage-all` to migrate the disks:

   ```
   sudo virsh migrate --offline --persistent KVM_Guest_Name \
       qemu+ssh://destination-host/system --copy-storage-all
   ```

4. Verify that the KVM guest now exists on the destination host.

   Run the following command and ensure that the migrated KVM guest appears in the list:

   ```
   sudo virsh list --all
   ```

   View basic information about the migrated KVM guest:

   ```
   sudo virsh dominfo KVM_Guest_Name
   ```

5. (Optional) If storage was copied then archive or remove old disk files on the source per the organization's retention policy.

5

KVM Instances: Hardware Configuration

Using the CLI, system administrators can add, remove, or change any of the following hardware configuration settings as required.

- Add Watchdog Device to KVM Instance
- Add vTPM Security to KVM Instance
- KVM Network Configuration
- KVM Storage Configuration
- KVM Memory and CPU Allocation Configuration

Add Watchdog Device to KVM Instance

Watchdog is an Oracle Linux service that runs in the background to monitor host availability and processes and reports back to the kernel.

The following information describes how to install the watchdog software package and enable its service. It also includes information about how to configure the watchdog daemon configuration file, and how to add watchdog settings to the XML configuration file for a KVM instance.

> **Note**
>
> Watchdog device configurations aren't supported on Arm-based KVMs. Arm-based KVMs are cloud-native virtual machines that operate on Arm-based processors.

What Do You Need?

- Administrator privileges.
- Existing KVM instance on host system.
  For details, see Create: KVM Instance.

Steps

Follow these steps to install and enable watchdog, update the watchdog daemon configuration as needed, and update the guest OS configuration file to include watchdog settings.

1. Install the Watchdog software package and enable the watchdog service on the guest OS.

   Example command syntax:

   ```
   sudo dnf install watchdog
   sudo systemctl enable --now watchdog.service
   ```

> **Note**
>
> The latest version of libvirt (9.x or later) includes a number of Watchdog enhancements and bug fixes over the earlier versions of libvirt.

The Watchdog service immediately starts and runs in the background.

2. Configure the watchdog service as needed for the guest OS.

   The `watchdog.conf` file includes all Watchdog configuration properties. For more details, see the `watchdog.conf(5)` Linux manual page.

3. Shut down the KVM instance.

   For details, see KVM: Shut Down Instance.

4. Add watchdog settings to the guest OS XML configuration file.

   a. Use the `virsh edit` command to edit the guest OS XML configuration.

      For example:

      ```
      virsh edit My_KVMGuest_Name
      ```

      > **Note**
      >
      > The `virsh edit` command opens the XML file in the text editor specified by the `$EDITOR` shell parameter. The **vi** editor is set by default.

   b. Update the guest OS XML configuration file to include the required watchdog device settings.

      For example:

      ```
      <devices>
           ...
           </input>
           <input type='mouse' bus='ps2'/>
           <input type='keyboard' bus='ps2'/>
           <watchdog model='i6300esb' action='poweroff'/>
           <graphics type='vnc' port='-1' autoport='yes'>
             <listen type='address'/>
           </graphics>
           ...
      </devices
      ```

      Where:

      - `model` – Specifies the emulated watchdog device driver. Valid property values are specific to the KVM machine type, for example:

Model Values	Description
i6300esb	The recommended device, which emulates an Intel 6300ESB.

Model Values	Description
ib700	Emulates an ISA iBase IB700, and is only compatible with the i440fx/pc machine type. **Note** This device doesn't work with the q35 machine type.

- `action` – (Optional) Describes the action taken when the watchdog timer expires.

Action Value	Description
reset	(Default option) Forcefully resets the guest VM.
shutdown	(Not recommended) Gracefully powers off the guest OS. **Important** The shutdown action requires that the guest is responsive to ACPI signals. In cases where the watchdog timer expired, the guests are typically unable to respond to ACPI signals. Therefore assigning a 'shutdown' action isn't recommended.
poweroff	Forcefully powers off the guest VM.
pause	Suspends the execution of the guest OS.
none	Does nothing.
dump	Automatically creates a dump file containing the core of the guest virtual machine so that it can be analyzed. **Note** To configure the directory to save the dump file, set the `auto_dump_path` in file `/etc/libvirt/qemu.conf`.
inject-nmi	Injects a non-maskable interrupt to the guest OS.

c. Save the guest OS XML configuration changes.

5. Start the KVM instance.

For details, see KVM: Start Instance.
The Watchdog service starts and runs immediately after a power reset.

Add vTPM Security to KVM Instance

The following provides information about the use of Virtual Trusted Platform Module (vTPM) security. It also includes configuration information for enabling vTPM security on a KVM instance.

About vTPM Security

A virtual Trusted Platform Module (vTPM) is a software-based representation of a physical Trusted Platform Module 2.0 chip. A vTPM acts as any other virtual device and provides security-related functions such as random number generation, attestation, and key generation. When added to a KVM instance, vTPM enables the guest OS to create and store keys that are private and not exposed to other guests. If a KVM instance is compromised and vTPM is enabled, the risk of its secrets being compromised is reduced because the keys are only usable to the KVM's guest OS for encryption or signing.

You can add a vTPM to an existing Oracle Linux 9 KVM. When you enable vTPM, the KVM files are encrypted but not the disks. Although, you can choose to add encryption explicitly for the KVM and its disks.

What Do You Need?

- Administrator privileges.
- Existing KVM instance on host system.
  For details, see Create: KVM Instance.

Steps

Follow these steps to install the vTPM software package and edit the guest OS configuration file to include vTPM security properties.

1. Install the vTPM software packages.

   ```
   sudo dnf install swtpm libtpms swtpm-tools
   ```

2. Shut down the KVM instance.

   For details, see KVM: Shut Down Instance.

3. Perform these steps to add the vTPM settings to the guest OS XML configuration file:

   a. Use the `virsh edit` command to edit the guest OS XML configuration.

      For example:

      ```
      virsh edit My_KVMGuest_Name
      ```

      > **Note**
      >
      > The `virsh edit` command opens the XML file in the text editor specified by the `$EDITOR` shell parameter. The **vi** editor is set by default.

   b. Update the guest OS XML configuration file to include the vTPM security properties.

For example:

```
<devices>
     ...
     </input>
     <input type='mouse' bus='ps2'/>
     <input type='keyboard' bus='ps2'/>
     <tpm model='tpm-crb'>
       <backend type='emulator' version='2.0'/>
     </tpm>
     <graphics type='vnc' port='-1' autoport='yes'>
       <listen type='address'/>
     </graphics>
     ...
</devices>
```

Where:

- model='tpm-crb' – sets the TPM model type as Command-Response Buffer (CRB).

  > **Note**
  >
  > The `tpm-crb` option is available only when you specify `version='2.0'`.

- type='emulator' – sets the device type as emulator.
- version='2.0' – sets the tpm version as 2.0.

> **Note**
>
> When creating a KVM instance for the first time on Oracle Linux 9, you can also use the `virt-install` command `--tpm` option to specify the TPM emulated device information at installation time. For example:
>
> ```
> virt-install --name MY_KVMGuest_ol8-tpm2 --memory 2048 --vcpus 2
> \
> --disk path=/systest/images/My_KVMGuest_ol8-tpm2.qcow2,size=20 \
> --location /systest/iso/ol8.iso --os-variant ol8 \
> --network network=default --graphics vnc,listen=0.0.0.0 --tpm
> emulator,model=tpm-crb,version=2.0
> ```

c. Save the guest OS XML configuration changes.

4. Start the KVM instance.

   For details, see KVM: Start Instance.

KVM Network Configuration

To configure and manage KVM virtual networks, see these topics:

- Overview: Virtual Networking
- Command Usage: Manage Virtual Network

Overview: Virtual Networking

Networking within a KVM environment is achieved by creating virtual Network Interface Cards (vNICs) on the KVM guest. vNICS are mapped to the host system's own network infrastructure in any of the following ways:

- Connecting to the virtual network running on the host.
- Directly using a physical interface on the host.
- Using Single Root I/O Virtualization (SR-IOV) capabilities on a PCIe device.
- Using a network bridge that enables a vNIC to share a physical network interface on the host.

vNICs are often defined when the KVM is first created, however the libvirt API can be used to add or remove vNICS as required, and because it can handle hot plugging, these actions can be performed on a running virtual machine without significant interruption.

Virtual Network Types:

A brief summary of the different types of virtual networks you can set up within a KVM environment are as follows:

- **Default Virtual Networking With NAT** – KVM networking can be complex because it involves: (1) physical components directly configured on the host system, (2) KVM configuration within `libvirt`, and (3) network configuration within the running guest OS. Therefore for many development and testing environments, it's often enough to configure vNICs to use the virtual network provided by `libvirt`. By default, the `libvirt` virtual network uses Network Address Translation (NAT) to enable KVM guests to gain access to external network resources. This approach is considered easier to configure and often facilitates similar network access already configured on the host system.
- **Bridged Network and Mapped Virtual Interfaces** – In cases where VMs might need to belong to specific subnetworks, a bridged network can be used. Network bridges use virtual interfaces that are mapped to and share a physical interface on the host. In this approach, network traffic from a KVM behaves as if it's coming from an independent system on the same physical network as the host system. Depending on the tools used, some manual changes to the host network configuration might be required before configuring it for KVM use.
- **Host Physical Network Interface** – Networking for VMs can also be configured to directly use a physical interface on the host system. This configuration can provide network behavior similar to using a bridged network interface in that the vNIC behaves as if it's connected to the physical network directly. Direct connections tend to use the `macvtap` driver to extend physical network interfaces to provide a range of functionality that can also provide a virtual bridge that behaves similarly to a bridged network but is considered easier to configure and maintain and more likely to offer improved performance.
- **Direct and Shared PCIe Passthrough** – Another KVM networking method is configuring PCIe passthrough where a PCIe interface supports the KVM network functionality. When using this method, administrators can choose to configure direct or shared PCIe passthrough networking. Direct PCIe passthrough allocates exclusive use of a PCIe device on the host system to a single KVM guest. Shared PCIe passthrough allocates shared use of an SR-IOV (Single Root I/O Virtualization) capable PCIe device to multiple KVM guests.

Both of these configuration methods require some hardware set up and configuration on the host system before attaching the PCIe device to a KVM guest(s) for network use.

KVM Tools for Configuring Virtual Network

In cases where network configurations are likely to be more complex, we recommend using Oracle Linux Virtualization Manager. The fundamental purpose of the CLI networking configurations and operations described in this guide is to facilitate the most basic KVM network deployment scenarios.

For details about Oracle Linux Virtualization Manager for more complex network configurations, see Oracle Linux Virtualization Manager documentation.

Command Usage: Manage Virtual Network

To manage virtual networks in a KVM environment, use the `virsh net-*` command. For example:

- `virsh net-list --all` – List all virtual networks configured on a host system.

  ```
  virsh net-list --all
  ```

  Output example:

  ```
   Name                 State      Autostart     Persistent
  ----------------------------------------------------------
   default              active     yes           yes
  ```

- `virsh net-info` – Display information about a network.

  ```
  virsh net-info default
  ```

  Output example:

  ```
  Name:           default
  UUID:           16318035-eed4-45b6-99f8-02f1ed0661d9
  Active:         yes
  Persistent:     yes
  Autostart:      yes
  Bridge:         virbr0
  ```

  Where:

  - `Name` = assigned network name.
  - `UUID` = assigned network identifier.
  - `virbr0` = virtual network bridge.

  > **Note**
  >
  > `virbr0` should not be confused with traditional bridge networking. In this case, the virtual bridge isn't connected to a physical interface. The virtual network bridge relies on NAT and IP forwarding to connect VMs to the physical network.

- `virsh net-dumpxml` – View the full configuration of a network.

  ```
  virsh net-dumpxml default
  ```

  Output example:

  ```
  <network>
    <name>default</name>
    <uuid>16318035-eed4-45b6-99f8-02f1ed0661d9</uuid>
    <forward mode='nat'>
      <nat>
        <port start='1024' end='65535'/>
      </nat>
    </forward>
    <bridge name='virbr0' stp='on' delay='0'/>
    <mac address='52:54:00:82:75:1d'/>
    <ip address='192.168.122.1' netmask='255.255.255.0'>
      <dhcp>
        <range start='192.168.122.2' end='192.168.122.254'/>
      </dhcp>
    </ip>
  </network>
  ```

  In this example, the virtual network uses a network bridge, called `virbr0`, not to be confused with traditional bridged networking. The virtual bridge isn't connected to a physical interface and relies on NAT and IP forwarding to connect VMs to the physical network beyond. `libvirt` also handles IP address assignment for VMs using DHCP. The default network is typically in the range 192.168.122.1/24.

- `virsh net-start` – Start an inactive, previously defined virtual network.

  ```
  sudo virsh net-start [--network] <network-identifier>
  ```

  Where: *network-identifier* stands for either network name or network UUID

- `virsh net-destroy` – Stop an active network and deallocate all resources used by it. For example, stopping appropriate dnsmasq process, releasing the bridge.

  ```
  sudo virsh net-destroy [--network] <network-identifier>
  ```

For a more complete list of `libvirt`'s network management commands, see the section 'Basic Command-line Usage for Virtual Networks' on the `libvirt` Virtual Networking site (https://wiki.libvirt.org/VirtualNetworking.html#virsh-xml-commands).

Command Usage: Add or Remove vNIC

You can use the `virsh attach-interface` command to add a new vNIC to an existing KVM. This command can be used to create a vNIC on a KVM that uses any of the networking types available in KVM.

```
virsh attach-interface --domain guest --type network --source default --config
```

You must specify the following parameters with this command:

- `--domain` – The KVM name, ID, or UUID.
- `--type` – The type of networking that the vNIC uses.
  Available options include:
  - `network` for a libvirt virtual network using NAT
  - `bridge` for a bridge device on the host
  - `direct` for a direct mapping to one of the host's network interfaces or bridges
  - `hostdev` for a passthrough connection using a PCI device on the host.
- `--source` – The source to be used for the network type specified.
  These values vary depending on the type:
  - For a `network`, specify the name of the virtual network.
  - For a `bridge`, specify the name of the bridge device.
  - For a `direct` connection, specify the name of the host's interface or bridge.
  - For a `hostdev` connection, specify the PCI address of the host's interface formatted as `domain:bus:slot.function`.
- `--config` – Changes the stored XML configuration for the guest VM and takes effect when the guest is started.
- `--live` – The guest VM must be running and the change takes place immediately, thus hot plugging the vNIC.
- `--current` – Affects the current guest VM.

More options are available to further customize the interface, such as setting the MAC address or configuring the target `macvtap` device when using some other network types. You can also use `--model` option to change the model of network interface that's presented to the VM. By default, the `virtio` model is used, but other models, such as `e1000` or `rtl8139` are available, Run `virsh help attach-interface` for more information, or see the `virsh(1)` manual page.

Remove a vNIC from a VM using the `virsh detach-interface` command. For example:

```
virsh detach-interface --domain guest --type network --mac 52:54:00:41:6a:65
--config
```

The `domain` or VM name and `type` are required parameters. If the VM has more than one vNIC attached, you must specify the `mac` parameter to provide the MAC address of the vNIC that you want to remove. You can obtain this value by listing the vNICs that are attached to a VM. For example, you can run:

```
virsh domiflist guest
```

Output similar to the following is displayed:

```
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet0      network    default    virtio      52:54:00:8c:d2:44
vnet1      network    default    virtio      52:54:00:41:6a:65
```

Bridged Networking: Setup

Using the CLI, administrators can set up a KVM bridged network with direct Virtual Network Interface Cards (vNICs). For more details, see these topics:

- Setup Guidelines: Bridged Network
- Create: Bridge Network Connection
- Bonded Interfaces for Increased Throughput

Setup Guidelines: Bridged Network

Traditional network bridging using Linux bridges is configurable by using the `virsh iface-bridge` command. With this command, administrators can create a bridge on a host system and add a physical interface to it. For example, the following command syntax creates a bridge named *vmbridge1* with the Ethernet port named *enp0s31f6*:

```
virsh iface-bridge vmbridge1 enp0s31f6
```

After establishing a bridged network interface, administrators can then attach it to a VM by using the `virsh attach-interface` command.

Traditional Linux Bridge Networking Complexities
Consider the following when using traditional Linux bridged networking for KVM guests:

- Setting up a software bridge on a wireless interface is considered complex because of the number of addresses available in 802.11 frames.
- The complexity of the code to handle software bridges can result in reduced throughput, increased latency, and additional configuration complexity.

Bridge Networking Advantages Using MacVTap Driver
The main advantage of a bridged network is that it lets the host system communicate across the network stack directly with any guests configured to use bridged networking.

Most of the issues related to using traditional Linux bridges can be easily overcome by using the `macvtap` driver which simplifies virtualized bridge networking. For most bridged network configurations in KVM, this is the preferred approach because it offers better performance and it's easier to configure. The `macvtap` driver is used when the network type in the KVM XML configuration file is set to `direct`. For example:

```
<interface type="direct">
  <mac address="#:##:##:##:#:##"/>
  <source dev="kvm-host-network-interface- name" mode="bridge"/>
  <model type="virtio"/>
  <driver name="vhost"/>
</interface>
```

Where:

- `mac address="#:##:##:##:#:##"` – The MAC address field is optional. If it is omitted, the `libvirt` daemon will generate a unique address.
- `interface type="direct"` – Used for MacVTap. Specifies a direct mapping to an existing KVM host device.

- `source dev="kvm-host-device" mode="bridge"` – Specifies the KVM host network interface name that will be used by the KVM guest's MacVTap interface. The **mode** keyword defines which MacVTap mode is used.

MacVTAP Driver Modes
The `macvtap` driver creates endpoint devices that follow the tun/tap ioctl interface model to extend an existing network interface so that KVM can use it to connect to the physical network interface directly to support different network functions. These functions can be controlled by setting a different mode for the interface. The following modes are available:

- `vepa` (Virtual Ethernet Port Aggregator) is the default mode and forces all data from a vNIC out of the physical interface to a network switch. If the switch supports a hairpin mode, different vNICs connected to the same physical interface can communicate through the switch. Many switches today don't support a hairpin mode, which means that virtual machines with direct connection interfaces running in VEPA mode are unable to communicate, but can connect to the external network by using the switch.
- `bridge` mode connects all vNICs directly to each other so that traffic between the virtual machines on same physical interface isn't sent to the switch but sent directly. The `bridge` mode option is the most useful for switches that don't support a hairpin mode, and when you need maximum performance for communications between VMs. Note the `bridge` mode, unlike a traditional software bridge, the host is unable to use this interface to communicate directly with the KVM.
- `private` mode behaves like a VEPA mode vNIC in the absence of a switch supporting a hairpin mode option. However, even if the switch does support the hairpin mode, two VMs connected to the same physical interface are unable to communicate with each other. This option supports limited use cases.
- `passthrough` mode attaches a physical interface device or an SR-IOV Virtual Function (VF) directly to the vNIC without losing the migration capability. All packets are sent directly to the configured network device. A one-to-one mapping exists between network devices and VMs when configured in `passthrough` mode because a network device can't be shared between VMs in this configuration.

> **Note**
>
> The `virsh attach-interface` command doesn't provide an option for you to specify the different modes available when attaching a direct type interface that uses the `macvtap` driver and defaults to vepa mode. The graphical `virt-manager` utility makes setting up bridged networks using `macvtap` easier and provides options for each different mode.

Create: Bridge Network Connection

The following information describes how to create and attach a virtual bridged network interface to a KVM guest using the MacVTap driver.
What Do You Need?

- Root privileges.
- An existing KVM guest on the host system.
- To use Ethernet devices as ports of the bridge, the physical or virtual Ethernet devices must be installed on the host system.

Steps

Follow these steps to configure a bridge network using the macvtap driver on an existing host KVM instance.

1. Create a bridge device and attach it to the physical network device interface on the host using the `virsh iface-bridge` command.

   Example:

   ```
   sudo virsh iface-bridge [bridge_name] [enp0s31f6]
   ```

   Where:

   - *`bridge_name`* – The name assigned to the bridge.
   - *`enp0s31f6`* – The physical interface the Ethernet port name used in this example.

2. Attach the bridge interface to the KVM instance using the `virsh attach-interface` command.

   Example:

   ```
   sudo virsh attach-interface --domain My_KVM_Guest_Name --type direct --
   source wlp4s0 --config
   ```

   Where:

   - *`My_KVM_Guest_Name`* – The name of the KVM instance.
   - `wlp4s0 --config` – The source interface name used in this example.

   For more details about using the `virsh attach-interface` command, see Command Usage: Add or Remove vNIC.

3. Shut down the KVM instance. For details, see KVM: Shut Down Instance

4. Edit the KVM XML configuration to set the source interface `mode` to `bridge`. For example:

   a. Use `virsh edit` to edit the file:

      ```
      sudo virsh edit [My_KVM_Guest_Name]
      ```

      > **Note**
      >
      > The `virsh edit` command opens the XML file in the text editor specified by the `$EDITOR` shell parameter. The **vi** editor is set by default.

   b. Set the source interface mode to `bridge`.

      > **Note**
      >
      > The source interface mode is set, by default, to `vepa`.

      For more details about how to set the source interface mode to `bridge`, see the `macvtap` driver example in Setup Guidelines: Bridged Network.

5. Save the KVM XML configuration changes.

6. Inform the `libvirt` daemon of the KVM XML configuration changes by using the `virsh undefine` and `virsh define` commands.

   Example:

   ```
   sudo virsh undefine [My_KVM_Guest_Name]
   sudo virsh define [My_KVM_Guest_Name-libvirt-xml-file]
   ```

   The `virsh undefine` command removes the existing KVM configuration and the `virsh define` replaces it with the updated configuration in the XML file.

7. Start the KVM instance. For details, see KVM: Start Instance.

   The direct network interface is attached in `bridge mode` and starts automatically when starting the KVM instance.

Bonded Interfaces for Increased Throughput

The use of bonded interfaces for increased network throughput is common when hosts might run several concurrent VMs that are providing multiple services at the same time. In this case, where a single physical interface might have provided enough bandwidth for applications hosted on a physical server, the increase in network traffic when running multiple VMs can have a negative impact on network performance when a single physical interface is shared. By using bonded interfaces, the KVM network throughput can significantly increase, thereby enabling you to take advantage of the high availability features available with network bonding.

Because the physical network interfaces that a VM might use are on the host and not on the VM, setting up any form of bonded networking for greater throughput or for high availability, must be configured on the host system. This process involves configuring network bonds on the host, and then attaching a virtual network interface such as a network bridge directly to the bonded network on the host.

To achieve high availability networking for any VMs, you must first configure a network bond on the host system. For details on how to set up network bonding, see *Network Bonding* in Oracle Linux 9: Setting Up Networking.

After the bond is configured, you can then configure the virtual machine network to use the bonded interface when you configure a network bridge. This can be done by either using: (1) the `bridge` mode for the interface type, or (2) a `direct` interface configured to use the `macvtap` driver's `bridge` mode. Note that the bonded interface can be used instead of a physical network interface when configuring the virtual network interface.

PCIe Passthrough: Setup

This section describes the following methods for configuring PCIe passthrough to KVM guests:

- **Direct PCIe Passthrough to KVM Guest Using `libvirt`**. Use this method to allocate exclusive use of a PCIe device on a host system to a single KVM guest. This method uses `libvirt` device assignment to configure a direct I/O path to a single KVM guest.

> **Note**
>
> Using direct PCIe passthrough can result in increased consumption of host system CPU resources and, thereby, decrease the overall performance of the host system.

For more information about configuring PCIe passthrough using this method, see Create: Direct PCIe Passthrough Connection.

- **Shared PCIe Passthrough to KVM Guests Using SR-IOV**. Use this method to allocate shared use of SR-IOV (Single Root I/O Virtualization) capable PCIe devices to multiple KVM guests. This method uses SR-IOV device assignment to configure a PCIe resource to be shared amongst several KVM guests. SR-IOV device assignment is beneficial in workloads with high packet rates or low latency requirements. For more information about SR-IOV PCIe passthrough, see the following topics:
  - Setup Guidelines: SR-IOV PCIe Passthrough
  - Create: SR-IOV PCIe Passthrough Connection
  - SR-IOV Enabled PCIe Devices

Create: Direct PCIe Passthrough Connection

The following information describes how to create a direct PCIe connection to a single KVM guest.

Exclusive PCIe Device Control

KVM guests can be configured to directly access the PCIe devices available on the host system and to have exclusive control over their capabilities. Use the `virsh` command to assign host PCIe devices to KVM guests. Note that after a PCIe device is assigned to a guest, the guest has exclusive access to the device and it's no longer available for use by the host or other guests on the system.

> **Note**
>
> The following procedure doesn't cover the configuration of enabling passthrough of SR-IOV Ethernet virtual devices. For instructions on how to configure passthrough for SR-IOV capable PCIe devices, see Create: SR-IOV PCIe Passthrough Connection.

Steps

Follow these steps to directly assign a host PCIe device to a KVM guest:

1. Shut down the KVM guest.

   ```
   sudo virsh shutdown GuestName
   ```

2. To identify the host attached PCIe devices and their assigned IDs, use the `lspci` command as follows:

   ```
   lspci -D|awk '{gsub("[:\\.]","_",$0); sub("^","pci_",$0); print;}'
   ```

   Where:

   - `lspci` lists all PCIe devices.
   - `-D` option lists the PCIe domain numbers for each device.
   - `awk` is a scripting language that manipulates the device IDs into a format usable by the `virsh` command.

For example, the output might look as follows:

```
pci_0000_00_00_0 Host bridge_ Intel Corporation 11th Gen Core Processor
Host Bridge/DRAM Registers (rev 01)
pci_0000_00_02_0 VGA compatible controller_ Intel Corporation TigerLake-LP
GT2 [Iris Xe Graphics] (rev 01)
pci_0000_00_04_0 Signal processing controller_ Intel Corporation TigerLake-
LP Dynamic Tuning Processor Participant (rev 01)
pci_0000_00_06_0 PCI bridge_ Intel Corporation 11th Gen Core Processor PCI
Express Controller (rev 01)
pci_0000_00_07_0 PCI bridge_ Intel Corporation Tiger Lake-LP Thunderbolt 4
PCI Express Root Port #0 (rev 01)
...
```

3. Select the device that you want to configure for passthrough and create a variable containing the device ID. For example:

   ```
   pci_dev="pci_0000_00_07_0"
   ```

4. Use the `virsh nodedev-dumpxml` command to calculate the PCIe device domain, bus, slot, and function parameters into usable variables. For example:

   ```
   domain=$(virsh nodedev-dumpxml $pci_dev --xpath '//domain/text()')
   bus=$(virsh nodedev-dumpxml $pci_dev --xpath '//bus/text()')
   slot=$(virsh nodedev-dumpxml $pci_dev --xpath '//slot/text()')
   function=$(virsh nodedev-dumpxml $pci_dev --xpath '//function/text()')
   ```

5. To identify the device source domain address required for passthrough, use the `print` function to convert the PCIe domain, bus, slot, and function variables to hexadecimal values.

   For example:

   ```
   printf "<address domain='0x%x' bus='0x%x' slot='0x%x' function='0x%x'/
   >\n" $domain $bus $slot $function
   ```

6. Assign the PCIe device to a KVM guest.

   Run `virsh edit`, specify the KVM guest name, and add the PCIe device domain address in the `<source>` section.
   For example:

   ```
   # virsh edit GuestName
   <hostdev mode='subsystem' type='pci' managed='yes'>
     <source>
        <address domain='0x0' bus='0x0' slot='0x14' function='0x3'/>
     </source>
   </hostdev>
   ```

> **Note**
>
> managed and unmanaged`libvirt` recognizes two management modes for handling PCIe devices: `managed='yes'` (default) or `managed='no"`. When the mode is set to `managed='yes'`, `libvirt` handles the unbinding of the device from the existing driver, resetting the device, and then binding it to the `vfio-pci` driver before starting the domain. In cases when the domain is stopped or the device is removed from the domain, `libvirt` unbinds it from the `vfio-pci` driver and rebinds it to the original driver. When the mode is set to `managed='no'`, you must manually detach the PCIe device from the host and then manually attach it to the `vfio-pci` driver.
> For example, to detach:
>
> ```
> sudo virsh nodedev-dettach pci_0000_device_ID_#
> ```
>
> To reattach:
>
> ```
> sudo virsh nodedev-reattach pci_0000_device_ID_#
> ```
>
> Alternatively, you can use Cockpit to attach and remove host devices. For more details, see *Add or Remove VM Host Devices* in the Oracle Linux: Using the Cockpit Web Console guide.

7. On the host system, enable guest management for virtual PCIe pass-through.

   ```
   sudo setsebool -P virt_use_sysfs 1
   ```

8. Start the KVM guest.

   ```
   sudo virsh start GuestName
   ```

   The PCIe device is successfully assigned to the KVM guest and the guest OS now has exclusive control over its capabilities.

Setup Guidelines: SR-IOV PCIe Passthrough

The Single Root I/O Virtualization (SR-IOV) specification is a standard for device assignment that can share a single PCIe resource among multiple KVM guests. SR-IOV provides the ability to partition a physical PCIe resource into virtual PCIe functions that can be discovered, managed, and configured as normal PCIe devices.

Passthrough configuration of PCIe devices using SR-IOV involves these functions:

- **Physical Functions (PF)** The physical function (PF) refers to the physical PCIe adapter device. Each physical PCIe adapter can have up to eight functions (although the most common case is one function). Each function has a full configuration space and is seen by software as a separate PCIe device. When the configuration space of a PCIe function includes SR-IOV support, then that function is considered an SR-IOV physical function. SR-IOV physical functions enable you to manage and configure SR-IOV settings for enabling virtualization and exposing virtual functions (VFs).
- **Virtual Function (VF)**. The virtual function (VF) refers to a virtualized instance of the PCIe device. Each VF is designed to move data in and out. VFs are derived from the physical

function (PF). For example, each VF is attached to an underlying PF and each PF can have from zero (0) to one (1) or more VFs. VFs have a reduced configuration space because they inherit most of their settings from the PF.

SR-IOV Advantages
Some key benefits for using SR-IOV for PCIe passthrough include:

- Optimized performance and capacity by enabling efficient sharing of PCIe resources.
- Reduced hardware costs through the creation of hundreds of VFs associated with a single PF.
- Dynamic control by the PF through registers designed to turn on the SR-IOV capability, eliminating the need for time-intensive integration.
- Increased performance through direct access to hardware from the virtual guest environment.

Create: SR-IOV PCIe Passthrough Connection

The following information describes how to create a SR-IOV PCIe passthrough connection for KVM guests.

SR-IOV Advantages and Capabilities
Single Root I/O Virtualization (SR-IOV) further extends Oracle Linux ability to operate as a high performance virtualization solution. With SR-IOV, Oracle Linux can assign virtual resources from PCI devices that have SR-IOV capabilities. These virtual resources known as virtual functions (VFs) appear as new assignable PCIe devices to KVM guests.

SR-IOV provides the same capabilities of assigning a physical PCI device to a guest. However, key benefits for using SR-IOV include optimization of I/O performance (as the guest OS interacts directly with device hardware), and the reduction of hardware costs (elimination for the need to manage a large system configuration of peripheral devices).

Steps

To configure SR-IOV PCIe passthrough to KVM guests, follow these steps:

1. Verify if the `Intel VT-d` or `AMD IOMMU` options are enabled in the system firmware at the BIOS/UEFI level. For more details, see the applicable Oracle server model documentation.
2. Verify if the `Intel VT-d` or `AMD IOMMU` options are activated in the kernel. If these kernel options haven't been enabled, perform the following.
   - For Intel virtualization, add the `intel_iommu=on` and `iommu=pt` parameters to the end of the `GRUB_CMDLINX_LINUX` line, within the quotes, in the `/etc/default/grub.cfg` file.

     > **Note**
     >
     > A symlink exists between `/etc/sysconfig/grub` and `/etc/default/grub`, therefore, you could alternatively choose to configure the /etc/sysconfig/grub.cfg file.

   - For AMD virtualization, add the `intel_iommu=on` and `iommu=pt` parameters to the end of the `GRUB_CMDLINX_LINUX` line, within the quotes, in the /etc/default/grub.cfg file. Regenerate `grub.cfg` file and then reboot the system for the changes to take affect.

     ```
     grub2-mkconfig -o /etc/grub.cfg
     ```

3. Use the `lspci` command to verify if an SR-IOV capable PCIe device is detected on the host system. For example:

   ```
   lspci -D|awk '{gsub("[:\\.]","_",$0); sub("^","pci_",$0); print;}'
   ```

   Where:

   - `lspci` lists all PCIe devices.
   - `-D` option lists the PCIe domain numbers for each device.
   - `awk` is a scripting language that manipulates the device IDs into a format usable by the `virsh` command.

   For example, the output might look as follows:

   ```
   pci_0000_00_00_0 Host bridge_ Intel Corporation 11th Gen Core Processor
   Host Bridge/DRAM Registers (rev 01)
   pci_0000_00_02_0 VGA compatible controller_ Intel Corporation TigerLake-LP
   GT2 [Iris Xe Graphics] (rev 01)
   pci_0000_00_04_0 Signal processing controller_ Intel Corporation TigerLake-
   LP Dynamic Tuning Processor Participant (rev 01)
   pci_0000_00_06_0 PCI bridge_ Intel Corporation 11th Gen Core Processor PCI
   Express Controller (rev 01)
   pci_0000_00_07_0 PCI bridge_ Intel Corporation Tiger Lake-LP Thunderbolt 4
   PCI Express Root Port #0 (rev 01)
   ...
   ```

   > **Note**
   >
   > For a list of SR-IOV compatible PCIe devices, see SR-IOV Enabled PCIe Devices .

4. Load the device driver kernel module.

   If an SR-IOV PCIe device is detected, the driver kernel module automatically loads.

   If required, you can pass parameters to the module using the `modprobe` command. The following example output shows the igb driver for an 82576 network interface card.

   ```
   sudo modprobe igb [<option>=<VAL1>,<VAL2>,]
   sudo lsmod |grep igb
   igb    82576  0
   dca    6708    1 igb
   ```

5. Activate the virtual functions (VFs) by performing the following:

   - **To set the maximum VFs offered by a kernel driver**, perform the following:

     a. To set the maximum VFs offered by a kernel driver, you must first remove the device driver kernel module. For example:

        ```
        sudo modprobe -r drivername
        ```

In the previous example in Step 4, `igb` is name of the driver. To find the device driver name, use the `ethtool` command. For example:

```
ethtool -i em1 | grep ^driver
```

b. Start the module with `max_vfs` set to 7 (or up to the maximum number allowed). For example:

```
sudo modprobe drivername max_vfs=7
```

c. Make the VFs persistent at boot.
Add the line `options drivername max_vfs=7` to any file in `/etc/modprobe.d`, for example:

```
sudo echo "options drivername max_vfs=7" >>/etc/modprobe.d/igb.conf
```

- **To allocate the required amount of VFs to create**, issue the following:

```
echo N > /sys/bus/pci/devices/${PF_DEV}/sriov_numvfs
```

Where:

– *N* is the number of VFs that you want the kernel driver to create.

– *${PF_DEV}* is the PCI bus/device/function ID for the physical device. For example: "0000:02:00.0" (as shown in the example output of Step 3.)

6. Use the `lspci | grep` command to list the newly added VFs.

For example, the following output lists VFs associated with the 82576 Network Controller.

```
sudo lspci | grep 82576
0b:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection (rev 01)
0b:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection(rev 01)
0b:10.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
0b:10.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
0b:10.2 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
0b:10.3 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
0b:10.4 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
0b:10.5 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
0b:10.6 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
0b:10.7 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
0b:11.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
0b:11.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
0b:11.2 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
```

```
01)
0b:11.3 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
0b:11.4 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
0b:11.5 Ethernet controller: Intel Corporation 82576 Virtual Function (rev
01)
```

The physical functions (PFs) correspond to `0b:00.0` and `0b:00.1` entries. Where all the VFs appear as a `Virtual Function` entry in the description.

7. Verify `libvirt` can detect the SR-IOV device by using the `virsh nodedev-list | grep` command.

   For the Intel 82576 network device example, the filtered output appears as follows:

```
virsh nodedev-list | grep 0b
pci_0000_0b_00_0
pci_0000_0b_00_1
pci_0000_0b_10_0
pci_0000_0b_10_1
pci_0000_0b_10_2
pci_0000_0b_10_3
pci_0000_0b_10_4
pci_0000_0b_10_5
pci_0000_0b_10_6
pci_0000_0b_11_7
pci_0000_0b_11_1
pci_0000_0b_11_2
pci_0000_0b_11_3
pci_0000_0b_11_4
pci_0000_0b_11_5
```

   Note that `libvirt` uses a similar notation to the `lspci` output. Punctuation characters, for example, such as a semicolon (`;`) and a period (`.`), appear in `lspci` output as underscores (_).

8. Use `virsh nodedev-dumpxml` command to review the SR-IOV physical and virtual functions device details.

   For example, advanced output shows details associated with the pci_0000_0b_00_0 physical function and its first corresponding virtual function (pci_0000_0b_10_0_),

```
sudo virsh nodedev-dumpxml pci_0000_0b_00_0
<device>
   <name>pci_0000_0b_00_0</name>
   <parent>pci_0000_00_01_0</parent>
   <driver>
      <name>igb</name>
   </driver>
   <capability type='pci'>
      <domain>0</domain>
      <bus>11</bus>
      <slot>0</slot>
      <function>0</function>
      <product id='0x10c9'>82576 Gigabit Network Connection</product>
```

```
      <vendor id='0x8086'>Intel Corporation</vendor>
   </capability>
</device>


sudo virsh nodedev-dumpxml pci_0000_0b_10_0
<device>
   <name>pci_0000_0b_10_0</name>
   <parent>pci_0000_00_01_0</parent>
   <driver>
      <name>igbvf</name>
   </driver>
   <capability type='pci'>
      <domain>0</domain>
      <bus>11</bus>
      <slot>16</slot>
      <function>0</function>
      <product id='0x10ca'>82576 Virtual Function</product>
      <vendor id='0x8086'>Intel Corporation</vendor>
   </capability>
</device>
```

Note the `bus`, `slot` and `function` parameters of the VF. These parameters are required in the next step to assign a VF to a KVM guest.

Copy these VF parameters into a temporary XML file, such as `/tmp/new-interface.xml` for example:

```
<interface type='hostdev' managed='yes'>
     <source>
       <address type='pci' domain='0' bus='11' slot='16' function='0'/>
     </source>
   </interface>
```

Note

- A MAC address is automatically generated if one isn't specified.
- The `<virtualport>` element is only used when connecting to an 802.11Qbh hardware switch.
- The `<vlan>` element transparently assigns a guest with a VLAN tagged *42*. When the KVM guest starts, it sees a network device of the type provided by the physical adapter, with the configured MAC address. This MAC address remains unchanged across host and guest reboots.

  The following `<interface>` example shows the syntax for the following optional elements: `<mac address>`, `<virtualport>`, and `<vlan>`. In practice, use either the `<vlan>` or `<virtualport>` element, but **not both simultaneously** as shown in the following example:

```
...
 <devices>
   ...
   <interface type='hostdev' managed='yes'>
     <source><address type='pci' domain='0' bus='11' slot='16'
function='0'/>
     </source><mac address='52:54:00:6d:90:02'><vlan><tag id='42'/>
     </vlan><virtualport type='802.1Qbh'>
     <parameters profileid='finance'/>
     </virtualport></interface>
   ...
 </devices>
```

9. Using the `new-interface.xml` file created in the previous step, and the `virsh attach-device` command, assign a VF of a SR-IOV PCIe device to a KVM guest.

   For example:

   ```
   virsh attach-device MyGuestName /tmp/new-interface.xml  --config
   ```

   The `--config` option ensures that the new VF is available after future restarts of KVM guest.

SR-IOV Enabled PCIe Devices

Note

Because of the continuous development of new SR-IOV PCIe devices and the Linux kernel, other SR-IOV capable PCIe devices might be available over time and aren't captured in the following table.

Table 5-1 PCIe Devices and Drivers

Device Name	Device Driver
Intel 82599ES 10 Gigabit Ethernet Controller	Intel `xgbe` Linux Base Drivers for Intel(R) Ethernet Network Connections For a list of the latest `xgbe`drivers, see http://e1000.sourceforge.net or http://downloadcenter.intel.com
Intel Ethernet Controller XL710 Series Intel Ethernet Network Adapter XXV710	Intel `i40e` Linux Base Drivers for Intel(R) Ethernet Network Connections For a list of the latest `i40e`drivers, see http://e1000.sourceforge.net or http://downloadcenter.intel.com
NVIDA (Mellanox) ConnectX-5, ConnectX-6 DX, and ConnectX-7	NVIDA (Mellanox) `mlx5_core` Driver
Intel 82576 Gigabit Ethernet Controller	Intel `igb` Linux* Base Drivers for Intel(R) Ethernet Network Connections For a list of the latest `xgbe`drivers, see http://e1000.sourceforge.net or http://downloadcenter.intel.com
Broadcom NetXtreme II BCM57810	Broadcom `bnx2x` Linux Base Drivers for Broadcom NetXtreme II Network Connections
Ethernet Controller E810-C for QSFP	Oracle Linux base driver packages available for Intel(R) Ethernet Network Connections
SFC9220 10/40G Ethernet Controller	`sfc` Linux base Driver
FastLinQ QL41000 Series 10/25/40/50GbE Controller	`qede` Poll Mode Driver for FastLinQ Ethernet Network Connections

KVM Storage Configuration

Libvirt handles various different storage mechanisms that you can configure for use by KVMs. These mechanisms are organized into different pools or units. By default, libvirt uses directory-based storage pools for the creation of new disks, but pools can be configured for different storage types including physical disk, NFS, and iSCSI.

Depending on the storage pool type that's configured, different storage volumes can be made available to any KVMs to be used as block devices. Sometimes, such as when using iSCSI pools, volumes don't need to be defined as the LUNs for the iSCSI target are automatically presented to the KVM.

Note that you don't need to define different storage pools and volumes to use libvirt with KVM. These tools help you to manage how storage is used and consumed by KVMs as they need it. You can use the default directory-based storage and take advantage of manually mounted storage at the default locations.

We recommend using Oracle Linux Virtualization Manager to easily manage and configure complex storage requirements for KVM environments. Alternatively, you can use Cockpit to manage KVM storage. For more details, see *Storage Management Tasks* in Oracle Linux: Using the Cockpit Web Console.

For more details on how to use the command line to manage storage configurations for KVM use, see these topics:

Storage Pools: Create and Manage

Storage pools provide logical groupings of storage types that are available to host the volumes that can be used as virtual disks by a set of VMs. A wide variety of different storage types are provided. Local storage can be used in the form of directory based storage pools, file system storage, and disk based storage. Other storage types such as NFS and iSCSI provide standard network based storage, while the RBD type provides distributed storage. More information is provided at https://libvirt.org/storage.html.

Storage pools help abstract underlying storage resources from the VM configurations. This abstraction is useful if you suspect that resources such as virtual disks might change physical location or media type. Abstraction becomes even more important when using network based storage because target paths, DNS, or IP addressing might change over time. By abstracting this configuration information, you can manage resources in a consolidated way without needing to update multiple KVM instances.

You can create transient storage pools that are available until the host reboots, or you can define persistent storage pools that are restored after a reboot.

Transient storage pools are started automatically as soon as they're created and the volumes that are within them are made available to VMs immediately, however any configuration information about a transient storage pool is lost after the pool is stopped, the host reboots, or if the libvirt daemons are restarted. The storage itself is unaffected, but VMs configured to use resources in a transient storage pool lose access to these resources. Transient storage pools are created using the `virsh pool-create` command.

For most use cases, consider creating persistent storage pools. Persistent storage pools are defined as a configuration entry that's stored within `/etc/libvirt`. Persistent storage pools can be stopped and started and can be configured to start when the host system boots. Libvirt can take care of automatically mounting and enabling access to network based resources when persistent storage is configured. Persistent storage pools are created using the `virsh pool-define` command, and usually need to be started after they have been created before you can use them.

For more details on how to use the command line to create and manage storage pools for KVM use, see these topics:

Creating a Storage Pool

Use the `virsh` tool to create a persistent storage pool.

1. Define the pool.

   ```
   virsh pool-define-as pool_name type
   ```

   Where:

   - *pool_name* – The name you assign to the pool.

- *type* – The storage type the pool uses.

See libvirt documentation for details about the storage types you can specify:

- Storage pool and volume XML format
- Storage Management

The following table provides examples of different pool types you can define:

Command	Configuration Details
`virsh pool-define-as pool_name dir \ --target /share/storage_pool`	Creates a pool with the name *pool_name* for a directory that's at */share/storage_pool* on the host system.
`virsh pool-define-as pool_name fs \ --source-dev /dev/sdc1 \ --target /share/storage_mount`	Creates file system based storage, that mounts a formatted block device, */dev/sdc1*, at the mount point */share/storage_mount*.
`virsh pool-define-as pool_name netfs \ --source-path /ISO \ --source-host nfs.example.com \ --target /share/storage_nfs`	Creates an NFS share as a storage pool.

2. Confirm the pool was defined.

   ```
   virsh pool-info pool_name
   ```

   You can also view a list of all pools on the system.

   ```
   virsh pool-list --all
   ```

3. If the target path doesn't exist, build the directory.

   ```
   virsh pool-build pool_name
   ```

4. Start the pool.

   ```
   virsh pool-start pool_name
   ```

5. Configure the pool to start automatically when the system boots.

   ```
   virsh pool-autostart pool_name
   ```

After you create a pool, you can create a storage volume within the pool. See Creating a Storage Volume for more information.

You can also indicate which pool to use when you create a VM using `virt-install`. Include the `--disk` argument and the `pool` and `size` sub options. For example:

```
virt-install
...
--disk pool=pool_name, size=80
```

Creating a Storage Pool from XML

Use the `virsh` tool to load a storage pool configuration from an XML file and create the pool.

1. Create an XML file with definitions for the storage pool.

   For more information on the XML format for a storage pool definition, see Storage pool and volume XML format.

   For example, you could create a storage pool for an iSCSI volume by creating an XML file named `pool_definition.xml` with the following content:

   ```
   <pool type='iscsi'>
     <name>pool_name</name>
     <source>
       <host name='192.0.2.1'/>
       <device path='iqn.2024-12.com.mycompany:my-iscsi-host'/>
     </source>
     <target>
       <path>/dev/disk/by-path</path>
     </target>
   </pool>
   ```

   The previous example assumes that an iSCSI server is already configured and running on a host with IP address *192.0.2.1* and that the iSCSI Qualified Name (IQN) is *iqn.2024-12.com.mycompany:my-iscsi-host*.

2. Run `virsh pool-define` to load the configuration information from the XML file into libvirt.

   For example, to load the `pool_definition.xml` file from the previous step, run:

   ```
   virsh pool-define pool_definition.xml
   ```

3. Confirm the pool was defined.

   ```
   virsh pool-info pool_name
   ```

   You can also view a list of all pools on the system.

   ```
   virsh pool-list --all
   ```

4. If the target path doesn't exist, build the directory.

   ```
   virsh pool-build pool_name
   ```

5. Start the pool.

   ```
   virsh pool-start pool_name
   ```

6. Configure the pool to start automatically when the system boots.

   ```
   virsh pool-autostart pool_name
   ```

Removing a Storage Pool

Use the `virsh` tool to stop and remove a persistent storage pool.

1. Stop the storage pool.

   ```
   virsh pool-destroy pool_name
   ```

2. Delete the directory of the storage pool.

   > **Note**
   >
   > The directory must be empty for this command to delete the directory.

   ```
   virsh pool-delete pool_name
   ```

3. Remove the storage pool definition from the system.

   ```
   virsh pool-undefine pool_name
   ```

4. Confirm the removal of the storage pool.

   ```
   virsh pool-list --all
   ```

Storage Volumes: Create and Manage

Storage volumes are created within a storage pool and represent the virtual disks that can be loaded as block devices within one or more VMs. Some storage pool types don't need storage volumes to be created individually as the storage mechanism might present these to VMs as block devices already. For example, iSCSI storage pools present the individual logical unit numbers (LUNs) for an iSCSI target as separate block devices.

Sometimes, such as when using directory or file system based storage pools, storage volumes are individually created for use as virtual disks. In these cases, several disk image formats can be used although some formats, such as `qcow2`, might require extra tools such as `qemu-img` for creation.

For disk based pools, standard partition type labels are used to represent individual volumes; while for pools based on the logical volume manager, the volumes themselves are presented individually within the pool.

Storage volumes can be sparsely allocated when they're created by setting the allocation value for the initial size of the volume to a value lower than the capacity of the volume. The allocation indicates the initial or current physical size of the volume, while the capacity indicates the size of the virtual disk as it's presented to the KVM. Sparse allocation is often used to over-subscribe physical disk space where KVMs might eventually require more disk space than is initially available. For a non-sparsely allocated volume, the allocation matches or exceeds the capacity of the volume. Exceeding the capacity of the disk provides space for metadata, if required.

You can use the `--pool` option if you have volumes with matching names in different pools on the same system and you need to specify the pool to use for any `virsh` volume operation. This practice is replicated across subsequent examples.

For more details on how to use the command line to create and manage storage volumes for KVM use, see these topics:

- Creating a Storage Volume
- Creating a Storage Volume from XML
- Cloning a Storage Volume
- Resizing a Storage Volume
- Deleting a Storage Volume

Creating a Storage Volume

Depending on the storage pool type, you can create a storage volume using the `virsh vol-create-as` command.

1. Run `virsh vol-create-as` and include the pool, volume name, and capacity as required arguments.

   For example:

   ```
   virsh vol-create-as \
   --pool pool_name \
   --name volume_name \
   --capacity 10G
   ```

   Many of the available options, such as the allocation or format have default values set, so you can typically only specify the name of the storage pool where the volume should be created, the name of the volume and the capacity that you require.

2. Verify the creation of the storage volume.

   ```
   virsh vol-info --pool pool_name volume_name
   ```

   Output similar to the following is displayed:

   ```
   Name:           volume_name
   Type:           file
   Capacity:       9.31 GiB
   Allocation:     8.00 GiB
   ```

Creating a Storage Volume from XML

Depending on the storage pool type, you can create a storage volume from an XML file using the `virsh vol-create` command. This command expects you to provide an XML file representation of the volume parameters.

1. Create an XML file where you define the storage volume.

The XML for a volume might depend on the pool type and the volume that's being created, but in the case of a sparsely allocated 10 GB image in `qcow2` format, the XML might look similar to the following:

```
<volume>
    <name>volume1</name>
    <allocation>0</allocation>
    <capacity unit="G">10</capacity>
    <target>
        <path>/home/testuser/.local/share/libvirt/images/volume1.qcow2</
path>
        <permissions>
            <owner>107</owner>
            <group>107</group>
                    <mode>0744</mode>
                    <label>virt_image_t</label>
              </permissions>
        </target>
</volume>
```

For more information, see Storage pool and volume XML format in the libvirt documentation.

2. Run `virsh vol-create` and include the pool and source XML file as required arguments.

   For example, to create a volume in storage pool named *pooldir* with an XML file named *volume1.xml*, run the following command:

   ```
   virsh vol-create pooldir volume1.xml
   ```

Cloning a Storage Volume

You can clone a storage volume using the `virsh vol-clone` command.

1. Run the `virsh vol-clone` command and include the name of the original volume and the name of the cloned volume as required arguments.

   For example:

   ```
   virsh vol-clone --pool pool_name volume1 volume1-clone
   ```

   The clone is created in the same storage pool with identical parameters.

2. Verify the creation of the cloned volume.

   ```
   virsh vol-list --pool pool_name --details
   ```

Resizing a Storage Volume

If a storage volume isn't being used by a VM, you can resize it by using the `virsh vol-resize` command.

- Run the `virsh vol-resize` command and provide the volume and capacity as required arguments.

For example:

```
virsh vol-resize --pool pool_name volume1 15G
```

> **Caution**
>
> Reducing the size of an existing volume can risk destroying data. However, if you need to resize a volume to reduce it, you must specify the `--shrink` option with the new size value.

Deleting a Storage Volume

You can delete a storage volume by running the `virsh vol-delete` command.

- Run `virsh vol-delete` and provide the volume name as a required argument.

  For example, to delete the volume named *volume1* in the storage pool named *pool_name*, run the following command:

  ```
  virsh vol-delete volume1 --pool pool_name
  ```

Virtual Disks: Create and Manage

Virtual disks are typically attached to VMs as block devices based on disk images stored at a given path. Virtual disks can be defined for a VM when it's created, or can be added to an existing VM.

> **Note**
>
> Command line tools available for managing virtual disks aren't completely consistent in terms of their handling of storage volumes and storage pools.

For more details about how to create and manage virtual disks for KVM use, see these topics:

- Attaching a Virtual Disk to an Existing VM
- Attaching a Virtual Disk when Creating a VM
- Detaching a Virtual Disk
- Resizing a Virtual Disk

Attaching a Virtual Disk to an Existing VM

You can use the `virsh attach-disk` command to attach a disk image to an existing VM. Command line tools to attach a volume to an existing VM are limited and GUI tools like `cockpit` are better suited for this operation. If you expect that you might need to work with volumes a lot, consider using Oracle Linux Virtualization Manager.

1. If the disk image is a volume, obtain its path by running the `virsh vol-list` command.

   ```
   virsh vol-list storage_pool_1
   ```

Output similar to the following is displayed:

```
 Name            Path
------------------------------------------------------------------------
 volume1         /share/disk-images/volume1.qcow2
```

2. Attach the disk image within the existing VM configuration so that it is persistent and attaches itself on each subsequent restart of the VM:

   ```
   virsh attach-disk --config \
   --domain guest_name \
   --source /share/disk-images/volume1.qcow2 \
   --target sdb1
   ```

   This command requires that you provide the path to the disk image when you attach it to the VM.

   You can use the following options:

   - `--live` – temporarily attach a disk image to a running VM.
   - `--persistent` – attach a disk image to a running VM and also update its configuration so that the disk is attached on each subsequent restart.

Attaching a Virtual Disk when Creating a VM

You can attach a storage volume to a VM as a virtual disk when the VM is created. The `virt-install` command enables you to specify the volume or storage pool directly for any use of the `--disk` option.

- Create a VM using `virt-install` and include the required `--disk` argument.

  To use an existing volume when creating a VM, include the `vol` option. For example:

  ```
  virt-install \
  --name guest \
  --disk vol=storage_pool/volume1.qcow2
  ...
  ```

  To create a virtual disk as a volume within an existing storage pool automatically at install, include the `pool` option. In this case, the `size` option is also required. For example:

  ```
  virt-install \
  --name guest \
  --disk pool=storage_pool,size=10
  ...
  ```

Detaching a Virtual Disk

You can remove a virtual disk from a VM by using the `virsh detach-disk` command.

> **⚠ Caution**
>
> Before you detach a disk from a running VM, ensure that you perform the appropriate actions within the guest OS to offline the disk correctly first. Otherwise, you might corrupt the file system. For example, unmount the disk in the guest OS so that it performs any sync operations that might still be remaining before you detach the disk.

1. Display a list of the block devices attached to a guest to identify the disk target.

   ```
   virsh domblklist guest_name
   ```

2. Detach the virtual disk.

   ```
   virsh detach-disk --config guest_name target_name
   ```

   You can use the following options:

   - `--live` – temporarily detach a disk image from a running KVM.
   - `--persistent` – detach a disk image from a running KVM and also update its configuration so that the disk is permanently detached from the KVM on subsequent restarts.

   Detaching a virtual disk from the VM doesn't delete the disk image file or volume from the host system. If you need to delete a virtual disk, you can either manually delete the source image file or delete the volume from the host.

   For example, to remove the disk at the target *sdb1* from the configuration for the KVM named *guest1*, you could run:

   ```
   virsh detach-disk --config guest1 sdb1
   ```

Resizing a Virtual Disk

You can resize a virtual disk image while a VM is running by using the `virsh blockresize` command.

1. Check the current size of all block devices attached to the VM.

   ```
   virsh domblkinfo guest_name --all --human
   ```

2. Find the path to the disk image and note the location.

   ```
   virsh domblklist guest_name --details
   ```

3. Run `virsh blockresize` and include the guest name, path to the disk, and intended size as required arguments.

   For example, to increase the size of the disk image at the source location */share/disk-images/volume1.qcow2* on the running VM named *guest1* to 20 GB, run:

   ```
   virsh blockresize guest_name /share/disk-images/volume1.qcow2 20GB
   ```

   The value you provide for size is a scaled integer which defaults to KiB if you omit a suffix.

The `virsh blockresize` command enables you to scale up a disk on a live VM, but it doesn't guarantee that the VM can immediately identify that the additional disk resource is available. For some guest operating systems, restarting the VM might be required before the guest can identify the additional resources available.

Individual partitions and file systems on the block device aren't scaled using this command. You need to perform these operations manually from within the guest, as required.

4. Verify that resizing has worked as expected by checking the block device information of the VM again.

```
virsh domblkinfo guest_name --all --human
```

KVM Memory and CPU Allocation Configuration

You can configure how many virtual CPUs (vCPUs) are active, and how much memory is available for each KVM instance. These hardware configuration changes can be made on a running KVM by hot plugging or hot unplugging; and the changes can be stored in the KVM's XML configuration file. Note that some changes can be limited by the KVM host, the hypervisor manufacturer, or by the original KVM configuration.

For more details on how to use the command line to configure memory and CPU allocation for KVM use, see these topics:

- Command Usage: Set Virtual CPU Count
- Command Usage: Allocate Memory

Command Usage: Set Virtual CPU Count

Optimizing vCPUs can impact the resource efficiency of any VMs. One way to optimize is to adjust how many vCPUs are assigned to a KVM instance. Hot plugging or hot unplugging vCPUs is when you configure vCPU count on a running KVM.

You can change the number of vCPUs that are active in a guest KVM using the `virsh setvcpus` command. By default, `virsh setvcpus` works on running guest KVMs. To change the number of vCPUs for a stopped KVM, add the `--config` option.

For example:

```
virsh setvcpus domain-name, id, or uuid count-value {--config | --live | --
current} --guest
```

Where:

- `setvcpus`: Sets the state of individual vCPUs using the hot(un)plug mechanism.

> **Note**
>
> The *count value* entered can't exceed the number of CPUs assigned to a KVM guest. Also, the allowable count value for vCPUs can vary depending on the following factors: host logical CPUs, hypervisor manufacturer, KVM guest OS, and so on.

- *domain-name*: A string value representing the KVM name, ID, or UUID.

- *count-value*: A number value representing the number of vCPUs.
- *--maximum*: Controls the maximum number of vCPUs that can be hot plugged the next time the guest KVM is booted. This option can only be used with the `--config` option.
- `--config`: Changes the stored XML configuration for the guest KVM and takes effect when the guest is started.
  - *--live*: The guest KVM must be running and the change takes place immediately, thus hot plugging a vCPU.
  - *--current*: Affects the current guest KVM.
- *--guest*: Sets the vCPU count directly in the running guest.

You can use the `--config` and `--live` options together if permitted by the hypervisor. If you don't specify `--config`, `--live`, or `--current`, the `--live` option is assumed. If you don't select an option and the guest KVM isn't running, the command fails. Furthermore, if no options are specified, it's up to the hypervisor whether the `--config` option is also assumed; and the hypervisor determines whether the XML configuration is adjusted to make the change persistent.

Command Usage: Allocate Memory

To improve the performance of a KVM, you can assign additional host RAM to a KVM instance. You can also decrease the amount of allocated memory to free up the resource for other KVMs or tasks. Hot plugging or hot unplugging memory is when you configure memory size on a running KVM.

You use the `virsh setmem` command to change the available memory for a KVM. To change the maximum memory that can be allocated, use the `virsh setmaxmem` command.

To change a KVM's memory allocation, run:

```
virsh setmem domain-name, id, or uuid --kilobytes size
```

You must specify the `size` as a scaled integer in kibibytes and the new value can't exceed the amount you specified for the KVM. Values lower than 64 MB are unlikely to work with most KVM guest operating systems. A higher maximum memory value doesn't affect active KVMs. If the new value is lower than the available memory, it shrinks memory usage possibly causing the KVM to crash.

Use following command options to allocate memory to a KVM instance:

- *`domain`*

  A string value representing the KVM name, ID, or UUID.

- *`size`*

  A number value representing the new memory size, as a scaled integer. The default unit is KiB, but you can select from other valid memory units:

  - b or bytes for bytes
  - KB for kilobytes (103 or blocks of 1,000 bytes)
  - k or KiB for kibibytes (210 or blocks of 1024 bytes)
  - MB for megabytes (106 or blocks of 1,000,000 bytes)
  - M or MiB for mebibytes (220 or blocks of 1,048,576 bytes)

  - GB for gigabytes (109 or blocks of 1,000,000,000 bytes)
  - G or GiB for gibibytes (230 or blocks of 1,073,741,824 bytes)
  - TB for terabytes (1012 or blocks of 1,000,000,000,000 bytes)
  - T or TiB for tebibytes (240 or blocks of 1,099,511,627,776 bytes)
- `--config`

  Changes the stored XML configuration for the guest KVM and takes effect when the guest is started.
- `--live`

  The guest KVM must be running and the change takes place immediately, thus hot plugging memory.
- `--current`

  Affects the memory on the current guest KVM.

To set the maximum memory that can be allocated to a KVM, run:

```
virsh setmaxmem domain-name_id_or_uuid size --current
```

You must specify the `size` as a scaled integer in kibibytes unless you also specify a supported memory unit, which are the same as for the `virsh setmem` command.

All other options for `virsh setmaxmem` are the same as for `virsh setmem` with one caveat. If you specify the `--live` option be aware that not all hypervisors support live changes to the maximum memory limit.

6

KVM Known Issues

The following topics describe known issues for Oracle Linux KVM. Note that when a workaround is available that information is also provided.

KVM Guest With vTPM Fails

Known Issue: KVM guests configured with vTPM can fail on Oracle Linux 9 when FIPS mode is enabled.
Description:

When FIPS mode is enabled on an Oracle Linux 9 host and a KVM is configured to use vTPM, the guest OS can fail to install or the KVM is unable to launch.

Workaround:
The current workaround is to disable FIPS mode if you need to run KVM guests with vTPM.

(Bug 34290427)

KVM Stack Upgrade Failure

Known Issue: KVM stack upgrade fails on Oracle Linux 9 when the `libvirt-lock-sanlock` package is installed.
Description:

The `libvirt-lock-sanlock` package has been renamed in the latest versions of QEMU 10.1/ libvirt 12.0. If you're running Oracle Linux 9 and upgrading from QEMU 7.2/libvirt 9.0 to QEMU 10.1/libvirt 12.0 and the `libvirt-lock-sanlock` package is installed, the upgrade fails.

Workaround:
Remove the existing `libvirt-lock-sanlock` package and install `libvirt-daemon-plugin-sanlock` before performing the upgrade:

```
sudo dnf remove libvirt-lock-sanlock
sudo dnf upgrade
sudo dnf install libvirt-daemon-plugin-sanlock
```

(Bug 39132071)

(aarch64) KVM Guest Installation Failure With `romfile` Error

When creating a KVM guest on an Arm platform, using the `virt-install` command, the installation can fail with the error:

```
failed to find romfile "efi-virtio.rom"
```

The efi-virtio.rom error is caused by `virt-install` generating a `<rom/>` element in the configuration without setting `bar="off"`, which QEMU can't resolve on this platform. Preferably, set this value for the PCI virtio network device when you run `virt-install`:

```
virt-install \
--name=vm-name \
... \
--host-device pci_0000_03_00_0 --config-xml='<rom bar="off"/>'
... \
```

For an existing VM, use `virsh edit` to update the XML configuration for the VM. Find the `<interface>` section for the network device and add `<rom bar='off'/>` as a new line within the `<interface>` block.

> **Note**
>
> This issue is specific to the QEMU version 10 and Libvirt version 12 Oracle Linux KVM stack.

(Bug 38642712)

(aarch64) KVM Guest Fails During UEFI PEI Boot Phase

During guest VM startup on an Arm platform, the guest might fail with an exception during the UEFI PEI boot phase:

```
...
PEIM Loaded: Tcg2ConfigPei.efi
PEIM Loaded: Tcg2Pei.efi
Synchronous Exception at 0x56E60
```

The issue is a CPU-level fault triggered by Tcg2Pei.efi trying to access an unmapped TPM MMIO register during firmware initialization.

To work around the issue, switch the firmware from `AAVMF_CODE.fd` to `AAVMF_CODE.pure-efi-notpm.fd`, which excludes the TPM PEI modules. Use the matching NVRAM template `AAVMF_VARS.pure-efi-notpm.fd`.

Preferably set these values when creating the VM using `virt-install`:

```
virt-install \
--name vm-name \
... \
```

```
loader=/usr/share/edk2/aarch64/AAVMF_CODE.pure-efi-notpm.fd, \
loader_ro=yes, \
loader_type=pflash, \
loader_secure=no, \
nvram=/usr/share/edk2/aarch64/AAVMF_VARS.pure-efi-notpm.fd \
... \
```

To update an existing VM configuration, use `virsh edit`:

```
...
<os>
<type arch="aarch64" machine="virt">hvm</type>
<loader readonly="yes" type="pflash">/usr/share/edk2/aarch64/AAVMF_CODE.pure-
efi-notpm.fd</loader>
<nvram template="/usr/share/edk2/aarch64/AAVMF_VARS.pure-efi-
notpm.fd">/var/lib/libvirt/qemu/nvram/ol_VARS.fd</nvram>
<boot dev="hd"/>
</os>
...
```

> **Note**
>
> This issue is specific to the EDK2/AAVMF package that's included in the Oracle Linux KVM stack.

(Bug 38642712)

(aarch64) KVM Guest NIC Hotplug Failure With `romfile` Error

When attaching a network device to a running KVM guest on an Arm platform, using the `virsh attach-device` command, the action can fail with the error:

```
failed to find romfile "efi-virtio.rom"
```

The efi-virtio.rom error is caused by `virsh attach-device` generating a `<rom/>` element in the configuration without setting `bar="off"`, which QEMU can't resolve on this platform. Use the `--rom bar=off` option when you attach a network device using `virsh attach-device`, for example:

```
virsh attach-interface vm_name --type bridge --source virbr0 --model virtio --
mac 52:54:00:96:82:11 --config --rom bar=off
```

> **Note**
>
> This issue is specific to the QEMU version 10 and Libvirt version 12 Oracle Linux KVM stack.

(Bug 39018293)