Oracle Linux

KVM User's Guide

F29966-23
May 2024

Oracle Linux KVM User's Guide,

F29966-23

Contents

3 KVM Usage

4 Known Issues for Oracle Linux KVM

Preface

Oracle Linux: 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.

Documentation License

Conventions

The following text conventions are used in this document:

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

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

About Oracle Linux KVM

This chapter provides a high-level overview of the Kernel-based Virtual Machine (KVM) feature on Oracle Linux, the user space tools that are available for installing and managing a standalone instance of KVM, and the differences between KVM usage in this mode and usage within a managed environment provided by Oracle Linux Virtualization Manager.

Description of the Oracle Linux KVM Feature

The KVM feature 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 7, Oracle Linux 8, and Oracle Linux 9 systems using either Red Hat Compatible Kernel (RHCK) or Unbreakable Enterprise Kernel (UEK).

By default, KVM is built into the kernel. KVM features are actively developed and might vary depending on platform and kernel release. If you're using UEK, see the release notes for the kernel release that you're using to obtain information about features and any known issues or limitations that might apply. See Unbreakable Enterprise Kernel documentation for more information.

For enterprise or clustered KVM deployments on Oracle Linux, consider using Oracle Linux Virtualization Manager which is a server virtualization management platform. Through its 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. Find out more at https://docs.oracle.com/en/virtualization/oracle-linux-virtualization-manager/.

Guest Operating System Requirements

The following guest operating systems can be used when installed within a standalone instance of KVM.

Linux Guest Operating Systems

Linux Operating System	32-bit Architecture	64-bit Architecture
Oracle Linux 6	Yes*	Yes
Oracle Linux 7	N/A	Yes
Oracle Linux 8	N/A	Yes
Oracle Linux 9	N/A	Yes
Red Hat Enterprise Linux 6	Yes*	Yes
Red Hat Enterprise Linux 7	N/A	Yes
Red Hat Enterprise Linux 8	N/A	Yes
Red Hat Enterprise Linux 9	N/A	Yes

Linux Operating System	32-bit Architecture	64-bit Architecture
CentOS 6	Yes*	Yes
CentOS 7	N/A	Yes
SUSE Linux Enterprise Server 12	N/A	Yes
SUSE Linux Enterprise Server 15	N/A	Yes
Ubuntu 16.04	N/A	Yes
Ubuntu 18.04	N/A	Yes
Ubuntu 20.04	N/A	Yes
Ubuntu 22.04	N/A	Yes

Important:

* `cloud-init` is unavailable for 32-bit architectures.

You can download Oracle Linux ISO images and disk images from Oracle Software Delivery Cloud: https://edelivery.oracle.com/linux.

Microsoft Windows Guest Operating Systems

Table 1-1 Microsoft Windows Supported Guest Operating Systems

Guest Operating System	64-bit	32-bit
Microsoft Windows Server 2022	Yes	N/A
Microsoft Windows Server 2019	Yes	N/A
Microsoft Windows Server 2016	Yes	N/A
Microsoft Windows Server 2012 R2	Yes	N/A
Microsoft Windows Server 2012	Yes	N/A
Microsoft Windows 11	Yes	Yes
Microsoft Windows 10	Yes	Yes
Microsoft Windows 8.1	Yes	Yes
Microsoft Windows 8	Yes	Yes

> **Caution:**
>
> Microsoft Windows 8 is no longer supported by Microsoft. See https://docs.microsoft.com/en-us/lifecycle/products/windows-8 for more information.
>
> Microsoft Windows 8.1 falls out of extended support by Microsoft in January 2023. See https://docs.microsoft.com/en-us/lifecycle/products/windows-81 for more information.

> **Note:**
>
> We recommend that you install the Oracle VirtIO Drivers for Microsoft Windows in Windows VMs for improved performance for network and block (disk) devices and to resolve common issues. The drivers are paravirtualized drivers for Microsoft Windows guests running on Oracle Linux KVM hypervisors.

Testing of all Microsoft Windows guests on KVM is performed 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 Operating System

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

Oracle Solaris 11.4.33 (Oracle Solaris 11.4 SRU 33) is the minimum version that provides VirtIO driver support.

For best results, follow these recommendations:

- 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.

You can download Oracle Solaris ISO images and disk images from Oracle Software Delivery Cloud: https://edelivery.oracle.com/.

System Requirements and Recommendations

Although most systems running Oracle Linux 7, Oracle Linux 8, or Oracle Linux 9 can use KVM, some general hardware recommendations, requirements, and guidelines should be followed to run a guest on a host system. Many of these depend on the kinds of applications being run on the virtual machine (VM) and the amount of work they're expected to perform.

- **Bare metal host**

  KVM can be used when it's run on a bare metal host. Nested virtualization scenarios aren't supported for KVM.

- **CPU**

The host system CPU must have virtualization features 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 operating system scale with the amount of physical memory available. For systems with lots of available physical memory, increase the reserved memory for the host operating system. For example, on a system with 1 TB memory, We recommend at least 20 GB available for the host operating system. If system work on a host and all VMs start 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, usually 6 GB, required for the host operating system should be met. Each VM requires its own storage for the guest operating system 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.

About Virtualization Packages

Oracle Linux provides several virtualization packages that enable you work with KVM. You can install virtualization packages from the Oracle Linux yum server or from the Unbreakable Linux Network (ULN). Packages are provided from various upstream projects, including:

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

The following packages are usually required for a virtualization host:

- `libvirt`: This package provides an interface to KVM, and the `libvirtd` daemon for managing guest VMs.
- `qemu-kvm`: This package installs the QEMU emulator that performs hardware virtualization so that guests can access host CPU and other resources.
- `virt-install`: This package provides command line utilities for creating and provisioning guest VMs.

- `virt-viewer`: This package provides a graphical utility that can be loaded into a desktop environment to access the graphical console of a guest VM.

Instead of installing virtualization packages individually, you can install virtualization package groups.

The `Virtualization Host` package group contains the minimum set of packages that are required for a virtualization host. If the Oracle Linux system includes a GUI environment, you can also choose to install the `Virtualization Client` package group.

Note that the Cockpit web console also provides a graphical interface to interact with KVM and `libvirtd` to set up and configure VMs on a system. See Oracle Linux: Using the Cockpit Web Console for more information.

2

Installing KVM User Space Packages

This chapter describes how to configure the appropriate ULN channels or yum repositories and how to install user space tools to manage a standalone instance of KVM. A final check is performed to validate whether the system can host guest VMs.

Configuring Yum Repositories and ULN Channels

Virtualization packages and their dependencies are available in various locations on the Oracle Linux yum server and on the Unbreakable Linux Network (ULN), depending on Oracle Linux release, the system architecture, and use case, or support requirements.

Oracle Linux 7

Due to the availability of several very different kernel versions and the requirement for more recent versions of user space tools that may break compatibility with RHCK, there are several different yum repositories and ULN channels across the different supported architectures for Oracle Linux 7. Packages in the different channels have different use cases and different levels of support. This section describes the available yum repositories and ULN channels for each architecture.

Repositories and Channels That Are Available for x86_64 Platforms

Yum Repositories	ULN Channels	Description
`ol7_latest`	`ol7_x86_64_latest`	The virtualization packages that are provided in this repository or ULN channel maximize compatibility with RHCK and with Red Hat Enterprise Linux. Packages from this repository or ULN channel are fully supported for all kernels.

Yum Repositories	ULN Channels	Description
ol7_kvm_utils	ol7_x86_64_kvm_utils	The virtualization packages that are provided in this repository or ULN channel take advantage of newer features and functionality available in upstream packages. These packages are also engineered to work with KVM features that are enabled in the latest releases of UEK. If you install these packages, you must also install the latest version of either UEK R4 or UEK R5.

Note:

The ol7_kvm_utils and ol7_x86_64_kvm_utils channels distribute 64-bit packages *only*. If you manually installed any 32-

Yum Repositories	ULN Channels	Description
		bit packages, for example, `libvirt-client`, yum updates from these channels will fail. To use the `ol7_kvm_utils` and `ol7_x86_64_kvm_utils` channels, you must first remove any 32-

Yum Repositories	ULN Channels	Description
		bit versions of the packages distributed by these channels that are installed on your system.
		You may choose to configure on-premises virtualization the same way that you configure systems on Oracle Cloud Infrastructure or other Oracle products that use KVM. Oracle Linux provides specific virtualization packages in this channel to assist with the configuration. Packages in this channel are delivered with limited support. Limited support coverage is only available for packages that are tested on Oracle Linux 7 with UEK. The following are the limitations and requirements: • A minimum of Oracle Linux 7.4 is required. • A minimum of Unbreakable Enterprise Kernel Release 4 is required. • Guest operating systems, as supported on Oracle

Yum Repositories	ULN Channels	Description
		Cloud Infrastructure and described at https://docs.oracle.com/iaas/Content/Compute/References/images.htm. • KVM guests boot by using iSCSI, VirtIO, VirtIO-SCSI or IDE device emulation.
ol7_developer ol7_developer_kvm_utils	ol7_x86_64_developer ol7_x86_64_developer_kvm_utils	The virtualization packages that are provided in these repositories or ULN channels take advantage of newer features and functionality that is available upstream, but are unsupported and are made available for developer use *only*. If you are using the Oracle Linux yum server, you can configure these repositories by installing the `oraclelinux-developer-release-el7` package and then enabling these repositories by editing the repository files or by using `yum-config-manager`.

Repositories and Channels That Are Available for aarch64 Platforms

Yum Repositories	ULN Channels	Description
ol7_latest	ol7_aarch64_latest	The virtualization packages that are provided in this repository or ULN channel include the latest virtualization packages, which are available and fully supported on Unbreakable Enterprise Kernel Release 5.
ol7_developer	ol7_aarch64_developer	The virtualization packages that are provided in this repository or ULN channel take advantage of newer features and functionality, which are available upstream, but are unsupported and are made available for developer use only.

> **Caution:**
>
> Virtualization packages may also be available in the `ol7_developer_EPEL` yum repository or the `ol7_arch_developer_EPEL` ULN channel. These packages are unsupported and contain features that might never be tested on Oracle Linux and may conflict with virtualization packages from other channels. If you intend to use packages from any of the repositories or channels that are previously listed, first uninstall any virtualization packages that installed from this repository. You can also disable this repository or channel or set exclusions to prevent virtualization packages from being installed from this repository.

Depending on your use case and support requirements, you must enable the repository or ULN channel that you require *before* installing the virtualization packages from that repository or ULN channel.

Subscribing to ULN Channels

If you're using ULN, follow these steps to ensure that the system is registered with ULN and that the appropriate channel is enabled:

1. Sign in to https://linux.oracle.com with your ULN username and password.
2. On the Systems tab, from the list of registered systems, select the link name for the specified system.
3. On the System Details page, select **Manage Subscriptions**.
4. On the System Summary page, from the list of available channels, select each of the required channels, then click the right arrow to move each channel to the list of subscribed channels.
5. Click Save Subscriptions.

Enabling Yum Repositories

If you're using the Oracle Linux yum server, you can either edit the repository configuration files in `/etc/yum.repos.d/` directly; or, if you have the `yum-utils` package installed, you can use the `yum-config-manager` command, for example:

```
sudo yum-config-manager --enable ol7_kvm_utils ol7_UEKR6
```

To prevent `yum` from installing the package versions from a particular repository, you can set an exclude option on these packages for that repository. For example, to prevent `yum` from installing the virtualization packages in the `ol7_developer_EPEL` repository, use the following command:

```
sudo yum-config-manager --setopt="ol7_developer_EPEL.exclude=libvirt* qemu*" --save
```

Oracle Linux 8

The number of options available on Oracle Linux 8 are significantly reduced as the available kernels are newer and there are less options from which to choose.

Repositories and Channels That Are Available for Oracle Linux 8

Yum Repositories	ULN Channels	Description
ol8_appstream	ol8_x86_64_appstream ol8_aarch64_appstream	The virtualization packages that are provided in this repository or ULN channel maximize compatibility with RHCK and with Red Hat Enterprise Linux. Packages from this repository or ULN channel are fully supported for all kernels. Packages released in this repository or ULN channel are released as part of the default DNF module: virt
ol8_kvm_appstream	ol8_x86_64_kvm_appstream ol8_aarch64_kvm_appstream	The virtualization packages that are provided in this repository or ULN channel take advantage of newer features and functionality available in upstream packages. These packages are also engineered to work with KVM features that are enabled in the latest releases of UEK. If you install these packages, you must also install the latest version of UEK R6 to use these features. The Oracle KVM stack packages released in this repository or ULN channel are available as a separate DNF module streams: virt:kvm_utils and virt:kvm_utils2. Additionally, some associated non-modular packages, such as virt-manager, edk2, swtpm and libtpms are available within this repository or channel. Packages that are included here are either not available in the standard AppStream repository or are available at a more recent version to take advantage of newer functionality. See Switching Application Streams on Oracle Linux 8 for more information.

Because the Application Stream repository or channel is required for system software on Oracle Linux 8, it's enabled by default on any Oracle Linux 8 system.

If you intend to use the virt:kvm_utils2 application stream for improved functionality and integration with newer features released within UEK, you must subscribe to the ol8_kvm_appstream yum repository or ol8_*base_arch*_kvm_utils ULN channel. Note that the virt:kvm_utils application stream is now a legacy stream on Oracle Linux 8.

Subscribing to ULN Channels

If you're using ULN, follow these steps to ensure that the system is registered with ULN and that the appropriate channel is enabled:

1. Sign in to https://linux.oracle.com with your ULN username and password.
2. On the Systems tab, from the list of registered systems, select the link name for the specified system.
3. On the System Details page, select **Manage Subscriptions**.
4. On the System Summary page, from the list of available channels, select each of the required channels, then click the right arrow to move each channel to the list of subscribed channels.
5. Click Save Subscriptions.

Enabling Yum Repositories

If you're using the Oracle Linux yum server, ensure that you have installed the most recent version of the `oraclelinux-release-el8` package and enable the required repositories. For example:

```
sudo dnf install -y oraclelinux-release-el8
sudo dnf config-manager --enable ol8_appstream ol8_kvm_appstream
```

Oracle Linux 9

The number of options available on Oracle Linux 9 are significantly reduced as the available kernels are newer and there are less options from which to choose. Note also that unlike Oracle Linux 8, the packages for Oracle Linux 9 aren't released as part of a DNF module.

Repositories and Channels That Are Available for Oracle Linux 9

Yum Repositories	ULN Channels	Description
`ol9_appstream`	`ol9_x86_64_appstream` `ol9_aarch64_appstream`	The virtualization packages that are provided in this repository or ULN channel maximize compatibility with RHCK and with Red Hat Enterprise Linux. Packages from this repository or ULN channel are fully supported for all kernels.

Yum Repositories	ULN Channels	Description
ol9_kvm_utils	ol9_x86_64_kvm_utils ol9_aarch64_kvm_utils	The virtualization packages that are provided in this repository or ULN channel take advantage of newer features and functionality available in upstream packages. These packages are also engineered to work with KVM features that are enabled in the latest releases of UEK. If you install these packages, you must also install the latest version of either UEK R7. **Note:** You must remove all existing virtualization packages before enabling this channel or repository.

Because the Application Stream repository or channel is required for system software on Oracle Linux 9, it's enabled by default on any Oracle Linux 9 system.

Subscribing to ULN Channels

If you're using ULN, follow these steps to ensure that the system is registered with ULN and that the appropriate channel is enabled:

1. Sign in to https://linux.oracle.com with your ULN username and password.

2. On the Systems tab, from the list of registered systems, select the link name for the specified system.
3. On the System Details page, select **Manage Subscriptions**.
4. On the System Summary page, from the list of available channels, select each of the required channels, then click the right arrow to move each channel to the list of subscribed channels.
5. Click Save Subscriptions.

Enabling Yum Repositories

If you're using the Oracle Linux yum server, ensure that you have installed the most recent version of the `oraclelinux-release-el9` package and enable the required repositories. For example:

```
sudo dnf install -y oraclelinux-release-el9
sudo dnf config-manager --enable ol9_kvm_utils ol9_UEKR7
```

Installing Virtualization Packages

Virtualization packages provide an interface to the KVM hypervisor, and user-space tools.

Installing Virtualization Packages During an Oracle Linux System Installation

You can use the following procedures to install virtualization packages during system installation. The Anaconda installation program can be used to install a single virtualization host. You can use a kickstart file to install virtualization hosts over the network.

Note that installation of virtualization software during system install on Oracle Linux 8 defaults to a KVM stack most compatible with RHCK. To use an alternate KVM stack you might need to perform steps to add other yum or dnf configuration and if you're running Oracle Linux 8 you might need to select an alternate application stream for the installation.

Using the Installation Program to Install Virtualization Hosts

The following steps describe how to install a virtualization host with the Oracle Linux graphical installation program:

1. Boot the Oracle Linux installation media and proceed to the **Software Selection** screen.
2. Select one of the following virtualization host types:
   - **Minimum Virtualization Host**

     (Available on Oracle Linux 7, Oracle Linux 8, and Oracle Linux 9)

     a. Select **Virtualization Host** in the **Base Environment** section.

     b. Select **Virtualization Host** in the **Add-ons for Selected Environment** section.
   - **Virtualization Host with GUI**

     (Not available on Oracle Linux 8)

     a. Select **Server with GUI** in the **Base Environment** section.

     b. Select the following package groups in the **Add-ons for Selected Environment** section:

- **Virtualization Client**
- **Virtualization Hypervisor**
- **Virtualization Tools**

3. Follow the prompts to complete the installation.

Using a Kickstart File to Install Virtualization Hosts

You can install virtualization hosts by specifying individual packages or package groups in the `%packages` section of a kickstart file.

Specify virtualization packages individually, as in the following example:

```
%packages
libvirt
qemu-kvm
virt-install
```

Specify the appropriate package groups for the installation type in the `%packages` section of the kickstart file by using the `@GroupID` format:

Minimum Virtualization Host

```
%packages
@virtualization-hypervisor
@virtualization-tools
# The following group is optional. Uncomment line to include...:
#@virtualization-platform
```

Virtualization Host with GUI

```
%packages
@virtualization-hypervisor
@virtualization-client
@virtualization-platform
@virtualization-tools
```

Installing Virtualization Packages on an Existing System

1. Log into the target Oracle Linux system with a user that has administrative privileges.
2. Ensure that the system has the appropriate yum repository or ULN channel enabled for the virtualization package versions that you want to install. See Configuring Yum Repositories and ULN Channels for more information.

> **Note:**
>
> If the target host system is running Oracle Linux 9 and you intend to use the virtualization packages available in `ol9_kvm_utils`. You must first remove any existing virtualization packages that might already be installed:
>
> a. Run the following command to remove packages:
>
> ```
> sudo dnf remove libvirt qemu-kvm edk2
> ```
>
> b. Enable the `ol9_kvm_utils` and `ol9_UEKR7` repositories:
>
> ```
> sudo dnf config-manager --enable ol9_kvm_utils ol9_UEKR7
> ```

3. Update the system so that it has the most recent packages available.
   - If you're using Oracle Linux 7, run the `yum update` command.
   - If you're using Oracle Linux 8 or Oracle Linux 9, run the `dnf update` command.
4. Install virtualization packages on the system.
   - If you're using Oracle Linux 7 run the following commands to install the base virtualization packages and other utilities:

     ```
     sudo yum groupinstall "Virtualization Host"
     sudo yum install qemu-kvm virt-install virt-viewer
     ```

   - If you're using Oracle Linux 8 run the following commands to install the base virtualization packages and other utilities:

     ```
     sudo dnf module install virt
     sudo dnf install virt-install virt-viewer
     ```

     See also Switching Application Streams on Oracle Linux 8.

   - If you're using Oracle Linux 9 run the following commands to install the base virtualization packages and other utilities:

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

     More steps are required to start virtualization services on Oracle Linux 9 after installation. For more details, see Validating the Host System.

Upgrading Virtualization Packages

Virtualization packages are updated by using the standard `yum update` or `dnf update` command. Note that to change the versions of the virtualization packages to match the versions that are shipped in a particular yum repository or ULN channel, you might need to specify the channel or repository from or to which you're installing packages. For example, you would update to the latest available virtualization packages that are available in the `ol7_kvm_utils` repository as follows:

```
sudo yum --disablerepo="*" --enablerepo="ol7_kvm_utils" update
```

To downgrade packages to a version in an alternate repository or channel, you must first remove the existing packages before installing the packages from the alternate repository. For example, to downgrade from the virtualization packages in the `ol7_kvm_utils` repository to the version of the same packages in the `ol7_latest` repository:

```
sudo yum remove libvirt* qemu* virt-install
sudo yum --disablerepo="*" --enablerepo="ol7_latest" install libvirt qemu-kvm virt-
install
```

Switching Application Streams on Oracle Linux 8

Virtualization packages on Oracle Linux 8 are released as a DNF module: `virt`. The default stream in the module contains packages that can work with both RHCK and UEK. Alternate versions of the packages that can take advantage of features that are only in UEK are available within a separate application stream, `virt:kvm_utils2`, that's provided along with some newer versions of non-modular packages within the `ol8_kvm_appstream` repository.

For more information about DNF modules and application streams, see Oracle Linux: Managing Software on Oracle Linux.

Switching to the Oracle KVM Stack

On an existing Oracle Linux 8 system, you can switch from the default KVM stack to the Oracle KVM stack in the `virt:kvm_utils2` stream by performing the following steps:

1. Remove any packages from the existing default `virt` stream:

   ```
   sudo dnf module remove virt -y --all
   ```

2. Reset the `virt` module state so that it's neither enabled nor disabled:

   ```
   sudo dnf module reset virt -y
   ```

3. Enable the `virt:kvm_utils2` module and stream:

   ```
   sudo dnf module enable virt:kvm_utils2 -y
   ```

4. Perform any necessary package upgrade or downgrade operations to handle dependencies for the enabled module and stream:

   ```
   sudo dnf --allowerasing distro-sync
   ```

5. Install the base packages from the `virt:kvm_utils2` stream:

   ```
   sudo dnf module install virt:kvm_utils2 -y
   ```

> **Caution:**
>
> Pre-existing guests that were created by using the default KVM stack might not be compatible and might not start using the Oracle KVM stack.

Note that although you can switch to the Oracle KVM stack and install the packages while using RHCK, the stack isn't compatible. You must be running a current version of UEK to use this software.

Switching to the Default KVM Stack

On an existing Oracle Linux 8 system, you can switch from the Oracle KVM stack to the default KVM stack by performing the following steps:

1. Remove any packages from the existing Oracle `virt:kvm_utils` or `virt:kvm_utils2` streams:

   ```
   sudo dnf module remove virt:kvm_utils -y --all
   sudo dnf module remove virt:kvm_utils2 -y --all
   ```

2. Reset the `virt` module state so that it's neither enabled nor disabled:

   ```
   sudo dnf module reset virt -y
   ```

3. Enable the `virt` module and stream:

   ```
   sudo dnf module enable virt -y
   ```

4. Perform any necessary package upgrade or downgrade operations to handle dependencies for the enabled module and stream:

   ```
   sudo dnf --allowerasing distro-sync
   ```

5. Install the base packages from the `virt` stream:

```
sudo dnf module install virt -y
```

> **Caution:**
>
> Pre-existing guests that were created by using the Oracle KVM stack aren't compatible and might not start using the default KVM stack.

Validating the Host System

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 functionalities, but KVM functionality is covered by testing the `qemu` virtualization type.

To test whether a system can act as a KVM host, run the following command:

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

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 such an option is available.

> **Note:**
>
> 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.

3
KVM Usage

Several tools exist for administering the `libvirt` interface with KVM. Usually, various different tools can perform the same operation. This document focuses on the tools that you can use from the command line. However, if you're using a desktop environment, you might consider using a graphical user interface (GUI), such as the VM Manager, to create and manage VMs. For more information about VM Manager, see https://virt-manager.org/.

The Cockpit web console also provides a graphical interface to interact with KVM and `libvirtd` to set up and configure VMs on a system. See Oracle Linux: Using the Cockpit Web Console for more information.

Checking the Libvirt Daemon Status

The `libvirt` daemon runs as a monolithic systemd service in Oracle Linux 7 and Oracle Linux 8. In Oracle Linux 9, the service is broken into multiple functional service sockets for more atomic control and logging for each virtualization component.

Oracle Linux 7 and Oracle Linux 8

To check the status of the `libvirt` daemon, run the following command on the virtualization host:

```
sudo systemctl status libvirtd
```

The output indicates whether the `libvirtd` daemon is running, as shown in the following example output:

```
 * libvirtd.service - Virtualization daemon
   Loaded: loaded (/usr/lib/systemd/system/libvirtd.service; enabled; vendor preset:
enabled)
   Active: active (running) since time_stamp; xh ago
```

If the daemon isn't running, start it by running the following command:

```
sudo systemctl start libvirtd
```

After you verify that the `libvirtd` service is running, you can start provisioning guest systems.

Oracle Linux 9

Individual `libvirt` functional components or drivers are modularized into separate daemons that are exposed using three systemd sockets for each driver.

The following systemd daemons are defined for individual drivers within `libvirt` for KVM:

- `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.

All the virtualization daemons must be running to expose the full virtualization functionality available in `libvirt`. A single service and three UNIX sockets are available for each daemon to expose different levels of access to the daemon. To enable all access levels and to start all daemons, run:

```
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.

To see the a list of all the sockets started and their current status, run:

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

More information on the modularization of the systemd `libvirt` daemon is available at https://libvirt.org/daemons.html

Working With Virtual Machines

A basic VM can be created without any complex storage, networking, CPU, or memory requirements. You can create a VM directly on the command line and you can start, stop, and remove it in the same way.

Creating a New Virtual Machine

The `virt-install` command is the most commonly used command line tool for creating and setting up new VMs. This utility has many options to enable you to customize a VM and control how it's created. For complete documentation on this tool, view the `virt-install(1)` manual page; or, for a quick list of options, you can run the `virt-install --help` command.

The following example, illustrates the creation of a basic VM and assumes that `virt-viewer` is installed and available to load the installer in a graphical environment:

```
virt-install --name guest-ol8 --memory 2048 --vcpus 2 \
--disk size=8 --location OracleLinux-R8.iso --os-variant ol8.0
```

The following are detailed descriptions of each of the options that are specified in the example:

- `--name` is used to specify a name for the VM. This name is registered as a domain within libvirt.
- `--memory` is used to specify the RAM available to the VM and is specified in MB.
- `--vcpus` is used to specify the number of virtual CPUs (vCPUs) that should be available to the VM.
- `--disk` is used to specify hard disk parameters. In this case, only the `size` is specified in GB. If a `path` isn't specified the disk image is created as a qcow file automatically. 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` is used to provide the path to the installation media. The location can be an ISO file, or an expanded installation resource hosted at a local path or remotely on an HTTP or NFS server.
- `--os-variant` is an optional specification but provides some default parameters for each VM that can help improve performance for a specific operating system or distribution. For a complete list of options available, run `osinfo-query os`.

When you run the command, the VM is created and automatically starts to boot using the install media specified in the `location` parameter. If you have the virt-viewer package installed and the command is run in a terminal within a desktop environment, the graphical console opens automatically and you can proceed with the guest operating system installation within the console.

Starting and Stopping Virtual Machines

After a VM is created within KVM, it's registered as a domain within libvirt and you can manage it by using the `virsh` command. To obtain a complete list of all registered domains and their status, run the following command:

```
virsh list --all
```

Output similar to the following is displayed:

```
 Id    Name                           State
----------------------------------------------------
 1     guest-ol8                      running
```

Use the `virsh help` command to view available options and syntax. For example, to find out more about the options available to listings of VMs, run `virsh help list`. This command shows options to view listings of VMs that are stopped or paused or that are active.

Starting a VM

To start a VM, run the following command:

```
virsh start guest-ol8
```

Output similar to the following is displayed:

```
Domain guest-ol8 started
```

Shutting Down a VM

To gracefully shut down a VM, run the following command:

```
virsh shutdown guest-ol8
```

Output similar to the following is displayed:

```
Domain guest-ol8 is being shutdown
```

Rebooting a VM

To reboot a VM, run the following command:

```
virsh reboot guest-ol8
```

Output similar to the following is displayed:

```
Domain guest-ol8 is being rebooted
```

Suspending a VM

To suspend a VM, run the following command:

```
virsh suspend guest-ol8
```

Output similar to the following is displayed:

```
Domain guest-ol8 suspended
```

Resuming a Suspended VM

To resume a suspended VM, run the following command:

```
virsh resume guest-ol8
```

Output similar to the following is displayed:

```
Domain guest-ol8 resumed
```

Forcefully Stopping a VM

To forcefully stop a VM, run the following command:

```
virsh destroy guest-ol8
```

Output similar to the following is displayed:

```
Domain guest-ol8 destroyed
```

Deleting a Virtual Machine

The following steps can be followed to remove a VM from a system:

1. Obtain information about the location of the VM by running the following command to dump information about the VM and check for the source files:

   ```
   virsh dumpxml --domain guest-ol8 | grep 'source file'
   ```

   The command returns output similar to the following:

   ```
   <source file='/home/testuser/.local/share/libvirt/images/guest-ol8-1.qcow2'/>
   ```

   This step is helpful if you're unsure of the path where the disk for the VM is located.
2. Shut down the VM, if possible, by running the following command:

   ```
   virsh shutdown guest-ol8
   ```

   If the VM can't be shut down gracefully you can force it to stop by running:

   ```
   virsh destroy guest-ol8
   ```
3. To delete the VM, run:

   ```
   virsh undefine guest-ol8
   ```

This step removes all configuration information about the VM from libvirt. Storage artifacts such as virtual disks remain intact. If you also need to remove these, you can delete them manually from their location returned in the first step in this procedure, for example:

```
rm /home/testuser/.local/share/libvirt/images/guest-ol8-1.qcow2
```

> **Note:**
>
> You can't delete a VM if it has snapshots. Remove any snapshots using the `virsh snapshot-delete` command before trying to remove a VM that has any snapshots defined.

Configuring a Virtual Machine With Watchdog Device

A virtual hardware Watchdog device configuration on a VM works with the guest OS to automatically trigger an action if the guest OS freezes or crashes. The `watchdog` software package must be installed on the guest VM and the service must be enabled. See *Configuring the Watchdog Service* in Oracle Linux 8: Managing Core System Configuration or in Oracle Linux 9: Managing Core System Configuration for more information.

> **Note:**
>
> Arm-based VMs do not support Watchdog device configurations.

To configure a virtual hardware Watchdog device on a guest Oracle Linux 8 or Oracle Linux 9 KVM VM, follow these steps:

1. Ensure that Watchdog is installed and the service is enabled on the guest OS.

   > **Note:**

   ```
   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.

2. Ensure that the Watchdog daemon is properly configured on the guest OS before adding the Watchdog device to the KVM VM configuration file.
   For details on how to configure the Watchdog daemon, see the `watchdog.conf(5)` manual page.
3. Shut down the KVM VM.
4. Edit the KVM VM configuration to include `watchdog` settings. You can either change the KVM VM XML directly, or you can use the `virsh edit` command to edit the XML and get validation for the changes:

- Use the `virsh edit` command to update the configuration for the VM:

  ```
  virsh edit guest-ol8
  ```

- Change the KVM VM's XML to include the watchdog device, as shown in the `watchdog` section in the following 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
  ```

  The following values are available for the model and action attributes that you can configure for the Watchdog device:

  – **model =** The required `model` attribute specifies which watchdog device driver is emulated. Note that the valid values are specific to the VM machine type.

Model Attribute	Description
i6300esb	The recommended device, which emulates an Intel 6300ESB.
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 =** The optional `action` attribute describes which action to take when the watchdog expires.

Action Attribute	Description
reset	Default action that forcefully resets the guest VM.

Action Attribute	Description
shutdown	Gracefully powers down the guest VM (not recommended). **Note:** The shutdown action requires that the guest is responsive to ACPI signals. In the sort of situations where the watchdog has expired, guests are usually unable to respond to ACPI signals. Therefore using 'shutdown' is not recommended.
poweroff	Forcefully powers off the guest VM.
pause	Pauses the execution of the guest VM.
none	Does nothing.
dump	Automatically dumps the guest VM. **Note:** The directory to save dump files can be configured by auto_dump_path in file `/etc/libvirt/qemu.conf`.
inject-nmi	Injects a non-maskable interrupt to the guest VM.

5. Save the XML file and restart the VM.

Configuring a Virtual Machine With a Virtual Trusted Platform Module

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, key generation. When added to a VM, a vTPM enables the guest operating system to create and store keys that are private and not exposed to the guest operating system. If a VM is compromised and vTPM is enabled, the risk of its secrets being compromised is reduced because the keys can be used only by the guest operating system for encryption or signing.

You can add a vTPM to an existing Oracle Linux 7, Oracle Linux 8, or Oracle Linux 9 KVM VM. When you configure a vTPM, the VM files are encrypted but not the disks. Although, you can choose to add encryption explicitly for the VM and its disks.

> **Note:**
>
> Virtual Trusted Platform Module is available on Oracle Linux 7, Oracle Linux 8, and Oracle Linux 9 KVM guests, but not on QEMU.

To provide a vTPM to an existing Oracle Linux 7, Oracle Linux 8, or Oracle Linux 9 KVM VM:

1. Install the vTPM packages:

   ```
   yum -y install swtpm libtpms swtpm-tools
   ```

2. Shut down the KVM VM.
3. Edit the KVM VM configuration to include TPM settings. You can either change the KVM VM XML directly, or you can use the `virsh edit` command to edit the XML and get validation for the changes:

   - Use the `virsh edit` command to update the configuration for the VM:

     ```
     virsh edit guest-ol8
     ```

   - Change the KVM VM's XML to include the TPM, as shown in the `tpm` section in the following 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>
     ```

   Note that if you're creating a new VM, the `virt-install` command on Oracle Linux 8 and Oracle Linux 9 also provides a `--tpm` option that enables you to specify the vTPM information at installation time, for example:

   ```
   virt-install --name guest-ol8-tpm2 --memory 2048 --vcpus 2 \
   --disk path=/systest/images/guest-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
   ```

   If you're using Oracle Linux 7, the `virt-install` command doesn't provide this option, but you can manually edit the configuration after the VM is created.

4. Start the KVM VM.

Configuring PCIe Passthrough to KVM Guests

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 Configuring Direct PCIe Passthrough for KVM Guests.

.

- **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:
  - Using SR-IOV for PCIe Passthrough to KVM Guests
  - Configuring SR-IOV PCIe Passthrough to KVM Guests
  - SR-IOV Enabled PCIe Devices

Configuring Direct PCIe Passthrough for KVM Guests

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 Configuring SR-IOV PCIe Passthrough to KVM Guests.

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.

Using SR-IOV for PCIe Passthrough to KVM Guests

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.

Configuring SR-IOV PCIe Passthrough to KVM Guests

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).

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 3-1 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

Working With Storage for KVM Guests

Libvirt handles various different storage mechanisms that you can configure for use by VMs. 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 VMs 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 VM.

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 VMs 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.

Storage Pools

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 RBD and Gluster types provide distributed storage mechanisms. 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 VM configurations.

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 libvirtd service is 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.

Creating a Storage Pool

To create a directory-based storage pool, `virsh pool-define-as` command with the `dir` subcommand. For example, you can create a pool with the name *pool_dir* for a directory that's at */share/storage_pool* on the host system:

```
virsh pool-define-as pool_dir dir --target /share/storage_pool
```

You can create other storage pool types by using the same `virsh pool-define-as` command. The options that you use with this command depend on the storage type that you select when you create a storage pool. For example, to create file system based storage, that mounts a formatted block device, */dev/sdc1*, at the mount point */share/storage_mount*, you can run:

```
virsh pool-create-as pool_fs fs --source-dev /dev/sdc1 --target /share/storage_mount
```

Similarly, you can add an NFS share as a storage pool, for example:

```
virsh pool-create-as pool_nfs netfs --source-path /ISO --source-host nfs.example.com \
--target /share/storage_nfs
```

You can also create an XML file representation of the storage pool configuration and load the configuration information from file using the `virsh pool-define` command. For example, you could create a storage pool for a Gluster volume by creating an XML file named `gluster_pool.xml` with the following content:

```
<pool type='gluster'>
  <name>pool_gluster</name>
  <source>
```

```
    <host name='192.0.2.1'/>
    <dir path='/'/>
    <name>gluster-vol1</name>
  </source>
</pool>
```

The previous example assumes that a Gluster server is already configured and running on a host with IP address *192.0.2.1* and that a volume named *gluster-vol1* is exported. Note that the `glusterfs-fuse` package must be installed on the host and verify that you can mount the Gluster volume before trying to use it with libvirt.

Run the following command to load the configuration information from the `gluster_pool.xml` file into libvirt:

```
virsh pool-define gluster_pool.xml
```

Note that we recommend using Oracle Linux Virtualization Manager when attempting to use complex network based storage such as Gluster.

For more information on the XML format for a storage pool definition, see https://libvirt.org/formatstorage.html#StoragePool.

Listing Storage Pools

You can list all the defined storage pools by using the `virsh pool-list` command, for example:

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

Use this command after you create a storage pool to verify that it the storage pool is available.

Starting a Storage Pool

To start a storage pool and make it accessible to any VMs, use the `virsh pool-start` command, for example:

```
virsh pool-start pool_dir
```

If you require the storage pool to also start at boot, run:

```
virsh pool-autostart pool_dir
```

Stopping a Storage Pool

To stop a storage pool use the `virsh pool-destroy` command, for example:

```
virsh pool-destroy pool_dir
```

Removing a Storage Pool

To remove the storage pool configuration completely use the `virsh pool-undefine` command, for example:

```
virsh pool-undefine pool_dir
```

Storage Volumes

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 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.

Note that 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 is presented to the VM. Sparse allocation is often used to over-subscribe physical disk space where VMs 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.

Note that 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.

Creating a New Storage Volume

Depending on the storage pool type, you can create new storage volumes using the `virsh vol-create` command. This command expects you to provide an XML file representation of the volume parameters. For example, to create a volume in storage pool named *pooldir* you could create an XML file, *volume1.xml* with the required parameters and run:

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

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 https://libvirt.org/formatstorage.html#StorageVol.

You can use the `virsh vol-create-as` command to create a volume by passing command line arguments to it. 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, for example:

```
virsh vol-create-as --pool pooldir --name volume1 --capacity 10G
```

Viewing Information About a Storage Volume

Use the `virsh vol-info` command to view information about a volume to determine its type, capacity, and allocation, for example:

```
virsh vol-info --pool pooldir volume1
```

Output similar to the following is displayed:

```
Name:           volume1
Type:           file
Capacity:       9.31 GiB
Allocation:     8.00 GiB
```

Cloning a Storage Volume

You can clone a storage volume using the `virsh vol-clone` command. This command takes the name of the original volume and the name of the cloned volume as a parameter and the clone is created in the same storage pool with identical parameters. For example:

```
virsh vol-clone --pool pooldir volume1 volume1-clone
```

Deleting a Storage Volume

You can delete a storage volume by running the `virsh vol-delete` command. For example, to delete the volume named *volume1* in the storage pool named *pooldir*, run the following command:

```
virsh vol-delete volume1 --pool pooldir
```

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. For example:

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

We don't advise reducing the size of an existing volume, as doing so 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.

Managing Virtual Disks

Virtual disks are attached to VMs, usually as block devices based on disk images stored at some or other path. Virtual disks can be defined for a VM when it's created, or can be added to an existing VM. The command line tools available for managing virtual disks aren't completely consistent in terms of their handling of storage volumes and storage pools.

Adding a Virtual Disk

Storage volumes can be attached 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. For example, to use an existing volume when creating a VM, using `virt-install`, specify the disk as follows:

```
virt-install --name guest --disk vol=storage_pool1/volume1.qcow2
...
```

You can equally use `virt-install` to create a virtual disk as a volume within an existing storage pool automatically at install. For example, to create a disk image as a volume within the storage pool named *storage_pool1*:

```
virt-install --name guest --disk pool=storage_pool1 size=10
...
```

Tools to attach a volume to an existing VM are limited and it's generally recommended that you use a GUI tool like `virt-manager` or `cockpit` to assist with this operation. If you expect that you might need to work with volumes a lot, consider using Oracle Linux Virtualization Manager.

You can use the `virsh attach-disk` command to attach a disk image to an existing VM. This command requires that you provide the path to the disk image when you attach it to the VM. If the disk image is a volume, you can obtain it's correct path by running the `virsh vol-list` command first.

```
virsh vol-list storage_pool_1
```

Output similar to the following is displayed:

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

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 guest1 \
 --source /share/disk-images/volume1.qcow2 --target sdb1
```

Note that you can use the `--live` option with this command to temporarily attach a disk image to a running VM; or you can use the `--persistent` option to attach a disk image to a running VM and also update it's configuration so that the disk is attached on each subsequent restart.

Removing a Virtual Disk

You can remove a virtual disk from a VM by using the `virsh detach-disk` command. For example, to remove the disk at the target *sdb1* from the configuration for the VM named *guest1*, you could run:

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

Note that you can use the `--live` option with this command to temporarily detach a disk image from a running VM; or you can use the `--persistent` option to detach a disk image from a running VM and also update it's configuration so that the disk is permanently detached from the VM on subsequent restarts. If 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. 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, or you might corrupt the file system.

Where disks are attached as block devices within a guest VM, you can obtain a listing of the block devices attached to a guest so that you can identify the disk target that's associated with a particular source image file, by running the `virsh domblklist` command, for example:

```
virsh domblklist guest1
```

Detaching a virtual disk from the VM does note 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.

Extending a Virtual Disk

You can extend a virtual disk image by using the `virsh blockresize` command while the VM is running. 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 guest1 /share/disk-images/volume1.qcow2 20GB
```

You can verify that the resize has worked by checking the block device information for the running VM, using the `virsh domblkinfo` command. For example to list all block devices attached to *guest1* in human readable format:

```
virsh domblkinfo guest1 --all --human
```

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 withing the guest, as required.

Working With Memory and CPU Allocation

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

Configuring 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 VM. Hot plugging or hot unplugging vCPUs is when you configure vCPU count on a running VM.

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

For example, run the following command to set the number of vCPUs on a running VM:

```
virsh setvcpus domain-name, id, or uuid count-value --live
```

Note that the count value can't exceed the number of CPUs assigned to the guest VM. The count value also might be limited by the host, hypervisor, or from the original description of the guest VM.

The following command options are available:

- *domain*

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

- *count*

  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 VM is booted. This option can only be used with the `--config` option.

- `--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 a vCPU.

- `--current`

  Affects the current guest VM.

- `--guest`

  Modifies the CPU state in the current guest VM.

- `--hotpluggable`

  Configures the vCPUs so they can be hot unplugged.

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 VM 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.

Configuring Memory Allocation

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

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

To change a VM'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 VM. Values lower than 64 MB are unlikely to work with most VM operating systems. A higher maximum memory value doesn't affect active VMs. If the new value is lower than the available memory, it shrinks possibly causing the VM to crash.

The following command options are available:

- *domain*

  A string value representing the VM 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 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 memory.

- `--current`

  Affects the memory on the current guest VM.

To set the maximum memory that can be allocated to a VM, 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 permit live changes of the maximum memory limit.

Setting Up Networking for KVM Guests

KVM provides tools to add or remove vNICs of different types and to help configure complex networking architectures. Networking in KVM is achieved by creating virtual Network Interface Cards (vNICs) on the guest VM. vNICS are mapped to the host system's own network infrastructure, by connecting to a virtual network running on the host itself; by directly using a physical interface on the host; using Single Root I/O Virtualization (SR-IOV) capabilities on a PCIe device; or by using a network bridge that enables the vNIC to share a physical network interface on the host.

vNICs are often defined when the VM is first created, however the libvirt API can be used to add or remove vNICS, as required, and also handles hot plugging to enable you to perform these actions on a running VM to avoid downtime.

Networking with KVM can be complex as it can involve components that are configured directly on the host itself, configuration for the VM within `libvirt` and also configuration for the network within the running guest operating system. Therefore for many development and testing environments, it's often enough to configure each vNIC to use the virtual networking provided by `libvirt`. This driver is used to create a virtual network that uses Network Address Translation (NAT) to enable VMs to gain access to external resources. This approach is simple to configure and often facilitates similar network access already configured on the host system.

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 configuration, network traffic from a VM 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 it can be set up for a VM.

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 which is easier to configure and maintain and which offers improved performance.

KVM can use SR-IOV for passthrough networking where a PCIe interface has this functionality. The SR-IOV hardware must be set up and configured on the host system before you can attach the device to a VM and configure the network to use this device.

Where network configuration is likely to be complex, we recommend using Oracle Linux Virtualization Manager. Simple networking configurations and operations are described here to facilitate most basic deployment scenarios.

Setting Up and Managing Virtual Networks

If you're considering using virtual networking with NAT for VM networking requirements, you can use the default virtual network that's set up by libvirt for VMs or you can create and manage different virtual networks within KVM to group VMs on their own subnetworks.

Use the following command to list all virtual networks that are configured on the host:

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

Output similar to the following is displayed:

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

You can find out more about a network using the `virsh net-info` command. For example, to find out about the default network, run:

```
virsh net-info default
```

Output similar to the following is displayed:

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

```
Persistent:     yes
Autostart:      yes
Bridge:         virbr0
```

Note that 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. To see the full configuration information about a network, use the `virsh net-dumpxml` command:

```
virsh net-dumpxml default
```

Output similar to the following is displayed:

```
<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>
```

Adding or Removing a vNIC

You can use the `virsh attach-interface` command to add a new vNIC to an existing VM. This command can be used to create a vNIC on a VM 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 VM 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 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
```

Note that 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 and Direct vNICs

Bridged vNICs enable a VM's network to act independently to the host's network configuration by sharing the same physical network interface to connect to the existing network infrastructure. This configuration can reduce complexity and is easy to manage.

Traditional network bridging using linux bridges is available using the `bridge` type when attaching an interface. The virsh iface-bridge command can be used to create a bridge on the host system and add a physical interface to it. For example, to create a bridge named *vmbridge1* with the Ethernet port named *enp0s31f6* attached, you can run:

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

After the bridge is created, you can attach it by using the `virsh attach-interface` command as described in Adding or Removing a vNIC.

Note that when using traditional linux bridged networking for KVM guests:

- It's not simple to set up a bridge on a wireless interface 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.

The main advantage that this approach offers, is that it allows the host system to 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 network. 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 is set to `direct`.

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 hairpin mode, different vNICs connected to the same physical interface are able to communicate via the switch. Many switches currently do not support hairpin mode, which means that VMs 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 VMs using the same physical interface isn't sent out to the switch and is facilitated directly. This mode is the most useful option when using switches that don't support hairpin mode, and when you need maximum performance for communications between VMs. Note that when configured in this mode, unlike a traditional software bridge, the host is unable to use this interface to communicate directly with the VM.
- `private` mode behaves a VEPA mode vNIC in the absence of a switch supporting hairpin mode. However, even if the switch does support hairpin mode, two VMs connected to the same physical interface are unable to communicate with each other. This option has 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.

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.

Nonetheless, it's not difficult to change the configuration of a VM by editing the XML definition for it directly. The following steps can be followed to configure a bridged network using the `macvtap` driver on an existing VM:

1. Attach a `direct` type interface to the VM using the `virsh attach-interface` command and specify the source for the physical interface to use for the bridge. In this example, the VM is called *guest1* and the physical network interface on the host is a wireless interface called *wlp4s0*:

   ```
   virsh attach-interface --domain guest1 --type direct --source wlp4s0 --config
   ```

2. Dump the XML for the VM configuration and copy it to a file that you can edit:

   ```
   virsh dumpxml guest1 > /tmp/guest1.xml
   ```

3. Edit the XML for the VM to change the `vepa` mode interface to use `bridged` mode. If many interfaces are connected to the VM, or you want to review changes, you can do this in a text editor. If you're happy to make this change globally, run:

   ```
   sed -i "s/mode='vepa'/mode='bridge'/g" /tmp/guest1.xml
   ```

4. Remove the existing configuration for this VM and replace it with the changed configuration in the XML file:

   ```
   virsh undefine guest1
   virsh define /tmp/guest1.xml
   ```

5. Restart the VM for the changes to take affect. The direct interface is attached in bridge mode and is persistent and automatically started when the VM boots.

Interface Bonding for Bridged Networks

The use of bonded interfaces for higher throughput is common where hosts might run several concurrent VMs that are providing multiple services at the same time. 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 where a single physical interface is shared. By using bonded interfaces, the throughput capability for VMs can be increased significantly and you can also take advantage of the high availability features that come with a network bond.

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, itself. This approach enables you to configure network bonds on the host and then to attach a virtual network interface, using a network bridge, directly to the bonded network on the host.

Network bonding of physical interfaces for Oracle Linux 7 is described in Oracle Linux 7: Setting Up Networking. For Oracle Linux 8, see Oracle Linux 8: Setting Up Networking. To achieve HA networking for any VMs, configure a network bond on the host system first.

When the bond is configured, configure the VM networks to use the bonded interface when you create a network bridge. You can do this by using either the `bridge` type interface or using a `direct` interface configured to use the `macvtap` driver's `bridge` mode. The bond interface can be used instead of a physical network interface when configuring a virtual network interface.

Cloning Virtual Machines

You can use two types of VM instances to create copies of VMs:

- **Clone**

  A clone is an instance of a single VM. You can use a clone to set up a network of identical VMs which you can optionally distribute to other destinations.

- **Template**

  A template is an instance of a VM that you can use as the cloning source. You can use a template to create multiple clones and optionally make modifications to each clone.

The difference between clones and templates is how they're used. For the created clone to work properly, ensure that you remove information and change configurations unique to the VM

that's being cloned before cloning. This information and configurations differs based on how you use the clones, for example:

- anything assigned to the VM such as the number of Network Interface Cards (NICs) and their MAC addresses.
- anything configured within the VM such as SSH keys.
- anything configured by an application installed on the VM such as activation codes and registration information.

You must remove some information and configurations from within the VM. Other information and configurations must be removed from the VM using the virtualization environment.

Preparing a Virtual Machine for Cloning

Before cloning a VM, you must prepare it by running the `virt-sysprep` utility on its disk image or by completing the following steps.

> **Note:**
>
> For more information on how to use the `virt-sysprep` utility to prepare a VM and understand the available options, see https://libguestfs.org/virt-sysprep.1.html.

1. Build the VM that you want to use for the clone or template.
   a. Install any needed software.
   b. Configure any non-unique operating system and application settings.
2. Remove any persistent or unique network configuration details.
   a. Run the following command to remove any persistent `udev` rules:

      ```
      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. 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 mustn't 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.

c. If you have `/etc/sysconfig/networking/profiles/default/ifcfg-eth[x]` and `/etc/sysconfig/networking/devices/ifcfg-eth[x]` files, ensure they have the same content as the `/etc/sysconfig/network-scripts/ifcfg-eth[x]` file.

> **Note:**
>
> Ensure that any other unique information is removed from the `ifcfg` files.

3. If the guest VM from which you want to create a clone is registered with ULN, you must de-register it. For more information, see the Oracle Linux: Unbreakable Linux Network User's Guide for Oracle Linux 6 and Oracle Linux 7.
4. Run the following command to remove any `sshd` public/private key pairs:

```
rm -rf /etc/ssh/ssh_host_*
```

> **Note:**
>
> Removing `ssh` keys prevents problems with `ssh` clients not trusting these hosts.

5. Remove any other application-specific identifiers or configurations that might cause conflicts if running on multiple machines.
6. Configure the VM to run the relevant configuration wizards the next time it boots.
   - For Oracle Linux 6 and below, run the following command to create an empty file on the root file system called `.unconfigured`:

     ```
     touch /.unconfigured
     ```

   - For Oracle Linux 7, run the following commands to enable the first boot and initial-setup wizards:

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

> **Note:**
>
> The wizards that run on the next boot depend on the configurations that have been removed from the VM. Also, on the first boot of the clone we recommend that you change the hostname.

> **Important:**
>
> Before proceeding with cloning, shut down the VM. You can clone a VM using `virt-clone` or `virt-manager`.

Cloning a Virtual Machine by Using the Virt-Clone Command

You can use `virt-clone` to clone VMs from the command line; however, you need root privileges for `virt-clone` to complete successfully. The `virt-clone` command provides several options that can be passed on the command line, which include general, storage configuration, networking configuration, and other miscellaneous options. Only the `--original` is required.

Run `virt-clone --help` to see a complete list of options, or see the `virt-clone(1)` manual page.

Run the following command to clone a VM on the default connection, automatically generating a new name and disk clone path:

```
virt-clone --original vm-name --auto-clone
```

Run the following command to clone a VM with multiple disks:

```
virt-clone --connect qemu:///system --original vm-name --name vm-clone-name \
--file /var/lib/libvirt/images/vm-clone-name.img --file /var/lib/libvirt/images/vm-clone-data.img
```

Cloning a Virtual Machine by Using Virtual Machine Manager

Complete the following steps to clone a guest VM using VM Manager.

1. Start VM Manager in one of the following ways:
   - Open VM Manager from the **System Tools** menu.
   - Run the `virt-manager` command as root.
2. From the list of guest VMs, right-click the guest VM you want to clone and click **Clone**.

   The **Clone VM** window opens.
3. In the **Name** field, change the name of the clone or accept the default name.
4. To change the **Networking** information, click **Details**. Then, enter a new MAC address for the clone and click **OK**.
5. For each disk in the cloned guest VM, select one of the following options:
   - **Clone this disk** - The disk is cloned for the cloned guest VM.
   - **Share disk with *guest-virtual-machine-name*** - The disk is shared by the guest VM to be cloned and its clone.
   - **Details** - Opens the **Change storage path** window to select a new path for the disk.
6. Click **Clone**.

4

Known Issues for Oracle Linux KVM

This chapter provides information about known issues for Oracle Linux KVM. If a workaround is available, that information is also provided.

Upgrading From QEMU 3.10 to Version 4.2.1 Can Prevent Existing KVM Guests From Starting on Oracle Linux 7

Attempting to upgrade a KVM host from QEMU version 3.10 to version 4.2.1 results in a `libvirt` server error that can prevent existing KVM guests from starting on an Oracle Linux 7 host.

An error similar to the following is displayed:

```
Upgrade qemu-3.1.0-7.el7.x86_64 to qemu-4.2.1-4.el7.x86_64, kvm can not be
started, got below libvirt service error:

Dec 21 15:10:48 ca-ex05db01.us.oracle.com libvirtd[23588]: Unable to read
from monitor: Connection reset by peer
Dec 21 15:10:48 ca-ex05db01.us.oracle.com libvirtd[23588]: internal error:
qemu unexpectedly closed the monitor: 2020-12-21T23:10:48.306929Z
qemu-system-x86_64: We need to set caching-mode=on for intel-iommu to enable
device assignment with IOMMU protection.
Dec 21 15:10:52 ca-ex05db01.us.oracle.com libvirtd[23588]: internal error:
Failed to autostart VM 'ca-ex05db01vm01.us.oracle.com': internal error: qemu
unexpectedly closed the monitor: 2020-12-21T23:10:48.306929Z
qemu-system-x86_64: We need to set caching-mode=on for intel-iommu to enable
device assignment with IOMMU protection.
Dec 21 15:10:52 ca-ex05db01.us.oracle.com libvirtd[23588]: nl_recv returned
with error: No buffer space available
```

To work around this issue so that KVM guests can run the updated qemu version, edit the XML file of each KVM guest, adding the `caching_mode='on'` parameter to the `iommu` section for each driver sub-element, as shown in the following example:

```
<iommu model='intel'>
      <driver aw_bits='48' caching_mode='on'/>
    </iommu>
```

(Bug ID 32312933)

Using vTPM With a Guest Fails on Oracle Linux 9 if FIPS Mode Is Enabled

If FIPS mode is enabled on an Oracle Linux 9 host and a VM is configured to use vTPM, the guest operating system fails to install or the VM is unable to launch. The current workaround is to disable FIPS mode if you need to run guests with vTPM.

(Bug 34290427)

Downgrading Application Streams Fail

Beginning with version 20744+e9607200 of the `virt:kvm_utils2` application stream, new packages were added. If you try to downgrade to a previous version of the `virt:kvm_utils2` application stream that does not contain the additional packages, the downgrade process fails with several package conflict error messages. This issue is the result of a limitation in DNF for handling dependencies in application streams.

To resolve this issue, you must remove the existing packages, reset the `virt:kvm_utils2` application stream, enable the older version of the `virt:kvm_utils2` application stream and then reinstall the packages. See Switching to the Oracle KVM Stack for steps to remove existing packages, resetting the application stream and then installing packages.

(Bug ID 34623368)