Oracle Linux

Podman User's Guide

F30921-28
December 2025

Oracle Linux Podman User's Guide,

F30921-28

Contents

Preface

Oracle Linux: Podman User's Guide describes how to use Podman, which is an open source, distributed-application platform that leverages Linux kernel technology to provide resource isolation management. Detail is provided on the advanced features of Podman and how it can be installed, configured, and used on Oracle Linux.

1

About Podman and Related Utilities

Podman, Buildah, and Skopeo are a set of tools to create, run, and manage applications across compatible Oracle Linux systems using Open Container Initiative compatible containers.

For information about the Open Container Initiative, see the upstream Open Container Initiative documentation.

About Podman

Podman provides a lightweight utility to run and manage Open Container Initiative compatible containers.

Podman deployments can reuse existing container images that are designed for Kubernetes, Docker, and Oracle Cloud Native Environment.

Podman is also intended as a drop-in replacement for Docker, so the CLI functions the same way if the `podman-docker` package is installed.

Similar to Docker, Podman integrates with Docker Hub, GitHub Container Registry, and Oracle Container Registry to share applications in a software-as-a-service (SaaS) cloud.

Unlike Docker, Podman doesn't require a running daemon to function. Containers run correctly on systems that are running either the latest Unbreakable Enterprise Kernel (UEK) release, or the Red Hat Compatible Kernel (RHCK). In addition, Podman containers can start and run without root permissions.

The Oracle Container Registry provides images for licensed commercial and open source Oracle software products. Images can also be used for development and testing purposes. The commercial license covers both production and testing use. The Oracle Container Registry provides a web interface for selecting Oracle images. If required, you must agree to terms of use before pulling the images by using the Podman client software. See Container Registries for more information about this service.

For more general information about Podman, see the upstream Podman documentation and the `podman(1)` manual page.

> **Note**
>
> Docker Hub and GitHub Container Registry aren't owned or maintained by Oracle, but Oracle supplies vendor-approved container images on those services for use with other software platforms and cloud providers.
>
> For more information, see the upstream Docker documentation and the upstream GitHub documentation.

About Buildah

Buildah is a utility for creating Open Container Initiative compatible container images.

Buildah provides a wider range of customization options than the more generic `podman build` command. If you create container images by using Buildah, you don't need a running daemon for the utility to function.

Buildah also doesn't cache builds by default. In addition, the utility can push container images to container registries, so it's suited for use with deployment scripts and automated build pipelines.

For more information, see Buildah.

About Skopeo

Skopeo is a utility for managing container images hosted on remote container registries.

Skopeo is useful for inspecting the contents of a container image without needing to first download it.

If container images are in a self-hosted (private) container registry, Skopeo can be used to seamlessly move container images from one location to another. Skopeo can also be used to bulk-delete container images.

For more information, see Skopeo.

2

Install Podman and Related Utilities

Install Podman and its related utilities such as Buildah and Skopeo on an Oracle Linux host.

To use Podman, the system must have the latest Unbreakable Enterprise Kernel (UEK) version or Red Hat Compatible Kernel (RHCK) installed.

Podman and the related utilities are available for Oracle Linux on the Oracle Linux yum server and the Unbreakable Linux Network (ULN).

The `container-tools` module (or metapackage, in the case of Oracle Linux 9 and Oracle Linux 10 hosts) is provided for convenience to install Podman and the utilities using a single command. However, Podman and its related utilities are designed to work independently of each other, so they can also be installed as individual packages. For example, Buildah has no dependency on Podman, so it's possible to separate the container build infrastructure from systems on which the containers are intended to run. You can optionally install Buildah on the same system that you run Podman, or you can install Buildah on another system. Similarly, you can install Skopeo separate from the other utilities according to specific requirements. To install these packages individually, see Installing Podman, Installing Buildah, and Installing Skopeo.

To install Podman and the related utilities on Oracle Linux 8 hosts, run the following command:

```
sudo dnf module install container-tools:ol8
```

To install Podman and the related utilities on Oracle Linux 9 and Oracle Linux 10 hosts, run the following command:

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

Podman, Buildah, Skopeo, and other related utilities are installed.

> **Tip**
>
> If you're using the Oracle Linux Cockpit Web console, you can install the `cockpit-podman` add-on application. This adds a **Podman containers** page to the Cockpit web console to monitor and manage Podman images, containers and pods. For example, the **Podman containers** page provides up-to-date container performance details, container CLI interaction ability, and options to create, run, and change container instances. For information on installing and using this add-on, see Oracle Linux: Using the Cockpit Web Console.

Installing Podman

Install standalone Podman without installing the associated utilities (Buildah or Skopeo) on an Oracle Linux host.

You can install either the `podman` package, or the `podman-docker` package. The `podman-docker` package effectively aliases the `docker` command to `podman`. The `podman-docker` package

might be helpful if you're more familiar with Docker, or where automation expects the `docker` command to be present. If you're installing the `podman-docker` package, replace this package name in the first step, and use the `docker` command instead of the `podman` command in the rest of this book.

1. Install Podman.

   ```
   sudo dnf install podman
   ```

2. Check the Podman configuration.

   Use the `podman info` command to display information about the configuration and version of Podman:

   ```
   podman info
   ```

3. Show the Podman commands.

   ```
   podman --help
   ```

   For more information, see the `podman(1)` manual page.

Installing Buildah

Install the Buildah utility without installing Podman or Skopeo on an Oracle Linux host.

1. Install Buildah.

   ```
   sudo dnf install buildah
   ```

2. Check the Buildah configuration.

   Use the `buildah info` command to display information about the configuration and version of Buildah:

   ```
   buildah info
   ```

3. Show the Buildah commands.

   ```
   buildah --help
   ```

   For more information, see the `buildah(1)` manual page.

Installing Skopeo

Install the Skopeo utility without installing Podman or Buildah on an Oracle Linux host.

1. Install Skopeo.

   ```
   sudo dnf install skopeo
   ```

2. Show the Skopeo commands.

   ```
   skopeo --help
   ```

For more information, see the `skopeo(1)` manual page.

3
Podman Images

Use Podman to search for, pull, review, and manage images that can be used to start containers and pods.

A container image is a read-only template that's used to generate a container. The image contains all the requirements for a service or application to run. Images can be limited in scope, for example, to host a single service such as a web server application. Or, images can be extensive enough to include a basic OS environment, such as a minimal Oracle Linux release.

Images can be tagged to identify different versions of the same image. Some images might include a default tag called `latest` so that Podman users can identify the most recent version of the image, but using those in production environments is considered bad practice because they might contain breaking changes or create unexpected variations between software deployments. Oracle Linux images don't provide a `latest` tag. For more information, see Oracle Linux Container Image Tagging Conventions.

Images are often hosted on container registries that can be accessed over HTTP/S by Podman instances to obtain particular image versions. Registries are described in more detail in Container Registries.

To change an existing image or create custom images, use the Buildah utility. For more information, see Buildah.

Searching for Images in Container Registries

Use the `podman search` command to run a search for container images in container registries that are configured on an Oracle Linux system.

For more information about the `podman search` command, use the `podman-search(1)` manual page.

For more information about how to configure container registries for use with Podman, see Container Registries.

Example 3-1 Search container registries for a container image

Search the configured registries for an `oraclelinux` image:

```
podman search oraclelinux
```

The output looks similar to:

```
NAME                                          DESCRIPTION
container-registry.oracle.com/os/oraclelinux  Oracle Linux
docker.io/library/oraclelinux                 Official Docker builds of
Oracle Linux.
docker.io/amd64/oraclelinux                   Official Docker builds of
Oracle Linux.
docker.io/litmusimage/oraclelinux
```

```
docker.io/arm64v8/oraclelinux                Official Docker builds of
Oracle Linux.
...
```

Pulling Images From a Container Registry

Pull a copy of a container image from a container registry using the `podman pull` command.

After you find an image, download a copy of it using the `podman pull` command, specifying the image reference as follows:

```
podman pull registry.host/repository/image_name:tag
```

- The *registry.host* domain is the resolvable hostname of the registry where the image is hosted, such as `example.com`. Although the registry host is often required, if the registry is already listed within the Podman configuration then specifying this value is optional.
- The *repository* value is optional and depends on how images are stored on the registry.
- The *image_name* value is required to specify which container image to download.
- The *tag* represents a version of the image and we recommend that this is specified. Many tools default to using the `latest` tag if no tag is specified but this can lead to errors and is considered bad practice. See Oracle Linux Container Image Tagging Conventions for more information on tags and why the `latest` tag is unreliable.

Shortcuts to registries and repositories for some commonly used container image names are stored in `/etc/containers/registries.conf.d/000-shortnames.conf`. Those shortcuts enable you to pull an image without needing to know the registry or repository to search, for example:

Container images are downloaded into the local container image store. This storage is described in more detail in Podman Storage.

For more information on the `podman pull` command, use the `podman-pull(1)` manual page.

Example 3-2 Pull an image from the Oracle Container Registry

Pull a slim Oracle Linux 9 container image from the Oracle Container Registry:

```
podman pull container-registry.oracle.com/os/oraclelinux:9-slim
```

The output looks similar to:

```
Trying to pull container-registry.oracle.com/os/oraclelinux:9-slim...
Getting image source signatures
Copying blob 60539f6b41ad done   |
Copying config 46cfa93e02 done   |
Writing manifest to image destination
46cfa93e021dd2ca65c70112d8d578484c3a9be71d4b05af6c027d4b7ae43182
```

Because the Oracle Container Registry is configured for use with Podman by default, the previous command could equally be specified as follows:

```
podman pull os/oraclelinux:9-slim
```

Example 3-3 Pull an image using a short name

Pull a slim Oracle Linux 9 container image from the Oracle Container Registry using the image short name:

```
podman pull oraclelinux:9-slim
```

Inspecting an Image

After downloading a container image, use the `podman inspect` command to review the container image metadata and default configuration settings.

For more information on the `podman inspect` command, use the `podman-inspect(1)` manual page.

Example 3-4 Inspect a container image

Review the slim Oracle Linux 9 container image:

```
podman inspect container-registry.oracle.com/os/oraclelinux:9-slim
```

The command provides similar JSON output and looks similar to the following:

```
[
     {
          "Id":
"46cfa93e021dd2ca65c70112d8d578484c3a9be71d4b05af6c027d4b7ae43182",
          "Digest":
"sha256:58c00a82b6a523256ecbeefc0f4dfdda11c460c1a2b5bf5ec1288fa0bb2fad68",
          "RepoTags": [
               "container-registry.oracle.com/os/oraclelinux:9-slim"
          ],
          "RepoDigests": [
               "container-registry.oracle.com/os/
oraclelinux@sha256:58c00a82b6a523256ecbeefc0f4dfdda11c460c1a2b5bf5ec1288fa0bb2
fad68",
               "container-registry.oracle.com/os/
oraclelinux@sha256:c7ba887b97ed69de05320f5b558c4dc42805194814935fa5ba329e6f638
4e06e"
          ],
          "Parent": "",
          "Comment": "",
          "Created": "2025-07-01T21:09:11.79180435Z",
          "Config": {
               "Env": [
                    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/bin"
               ],
               "Cmd": [
                    "/bin/bash"
               ],
               "Labels": {
                    "io.buildah.version": "1.33.11"
               }
          },
```

```
          "Version": "",
          "Author": "",
          "Architecture": "amd64",
          "Os": "linux",
          "Size": 116893435,
          "VirtualSize": 116893435,
          "GraphDriver": {
               "Name": "overlay",
               "Data": {
                    "UpperDir": "/home/opc/.local/share/containers/storage/
overlay/d62ccb1665e5b7444fa53147fd18e2a7989fff5f330480711e62445a6443ca92/
diff",
                    "WorkDir": "/home/opc/.local/share/containers/storage/
overlay/d62ccb1665e5b7444fa53147fd18e2a7989fff5f330480711e62445a6443ca92/work"
               }
          },
          "RootFS": {
               "Type": "layers",
               "Layers": [

"sha256:d62ccb1665e5b7444fa53147fd18e2a7989fff5f330480711e62445a6443ca92"
               ]
          },
          "Labels": {
               "io.buildah.version": "1.33.11"
          },
          "Annotations": {},
          "ManifestType": "application/
vnd.docker.distribution.manifest.v2+json",
          "User": "",
          "History": [
               {
                    "created": "2025-07-01T21:09:10.947182529Z",
                    "created_by": "/bin/sh -c #(nop) ADD
file:6bff5a6139bdf95b00802c939bfcb5e9b8c2324a854e6124b95d3312f3428158 in / "
               },
               {
                    "created": "2025-07-01T21:09:11.791909449Z",
                    "created_by": "/bin/sh -c #(nop) CMD [\"/bin/bash\"]",
                    "comment": "FROM 45b3741d0ae8",
                    "empty_layer": true
               }
          ],
          "NamesHistory": [
               "container-registry.oracle.com/os/oraclelinux:9-slim"
          ]
     }
]
```


Listing Local Images

Use the `podman images` command to review a list of locally stored container images.

For more information on the `podman images` command, use the `podman-images(1)` manual page.

Example 3-5 List local container images

List all the locally stored container images that have already been downloaded from a container registry

```
podman images
```

The command provides similar output similar to the following:

```
REPOSITORY                                      TAG         IMAGE ID
CREATED      SIZE
container-registry.oracle.com/os/oraclelinux  9-slim      46cfa93e021d  5
days ago   117 MB
```

Deleting a Local Image

Delete a locally stored container image using the `podman rmi` command.

Container images can't be deleted if they're still in use by a container, even if that container isn't running. You must remove all the containers that depend on a container image before you can remove that container image.

For more information on the `podman rmi` command, use the `podman-rmi(1)` manual page.

Example 3-6 Delete a local container image

Delete an image named `oraclelinux:9-slim` that's stored locally:

```
podman rmi oraclelinux:9-slim
```

The command provides output similar to the following:

```
Untagged: container-registry.oracle.com/os/oraclelinux:9-slim
Deleted: 46cfa93e021dd2ca65c70112d8d578484c3a9be71d4b05af6c027d4b7ae43182
```

4

Podman Containers

Use Podman to create, run, and manage containers.

Containers are running instances of images. Each container uses an image as its starting point and then loads into run time by using the parameters that are provided when it's created or run.

Podman can run containers based on images that comply with the Open Container Initiative specification. Most Podman commands map directly to the command equivalents that are available in the Docker CLI.

A key difference between Podman and Docker is that while the Docker Engine runs as a service on the host and all actions are performed by the service, Podman runs as a standalone runtime so that each operation is independent. This difference is important, because it changes the security model around working with images and containers.

As Podman operations aren't dependent on a service daemon running as a particular user on the system, Podman provides more isolated containers than Docker. This also means that you can either run Podman with and without root permissions.

Podman respects user namespaces, so several users on a single host can run their own containers and local image stores without conflicts. Because containers running within a user's namespace are limited to the permissions available to that user on the host system, Podman can be more secure in some scenarios than Docker.

When running Podman as a standard user without root permissions, functionality can be more limited. For example, most container networking is achieved by using port mapping and port forwarding, and workarounds are required to make that functionality available to standard users without compromising broader system security. Many such limitations that arise for a standard user can be mitigated by using the provided network drivers and running groups of containers inside a pod. For more information about networking and Podman, see Podman Networking. For more information about pods, see Podman Pods.

In general, the instructions provided here apply similarly regardless of whether Podman is run by a system administrator or not.

Creating Containers

Create a container from a container image using the `podman create` command.

You can create a container from an existing image using the `podman create` command. This creates, but doesn't start a container.

If the image doesn't already exist on the local system, Podman searches the remote registries for a matching image and pulls the image automatically.

The container is created and the container ID is displayed in the output. The container is created with the initial state of `Created`.

You can specify other options when creating a container, such as whether it belongs to a particular pod or whether it uses a particular network or port mapping. Use the `podman help create` command to see more information. Options are extensive and can be used to apply a wide range of runtime functionality to any container.

For more information on the `podman create` command, use the `podman-create(1)` manual page.

Example 4-1 Create a container from an image

Create a container named `oracle` from an `oraclelinux:9-slim` image:

```
podman create --name oracle oraclelinux:9-slim
```

Show the container is created by listing all the containers:

```
podman ps --all
```

The looks similar to:

```
CONTAINER ID  IMAGE
COMMAND     CREATED         STATUS      PORTS       NAMES
4fbb572f7dcb  container-registry.oracle.com/os/oraclelinux:9-slim  /bin/
bash   1 minute ago    Created                 oracle
```

Running Containers

Podman offers many ways to run a container. The examples here show you a few options.

Use the `podman run` command to run a container. If you run a container that doesn't already exist, it's created automatically. If the image that the container uses isn't available locally, Podman searches the remote registries for a matching image and pulls the image automatically.

You can create a container and connect to it in a single step by using the `-it` flag. The `-i` flag makes the container interactive and `-t` connects the local terminal to the container. This flag combination is commonly used in conjunction when running a specified shell as part of the `podman run` command.

You can start a container, and run a single command. When the command is completed, the container is destroyed. Use the `podman run` command with the `--rm` flag to do this.

You can also create Podman containers that continue to run as a background daemon by including the `-d` flag in the command.

If a container runs a shell as the primary process (PID 1) and you intend to detach it, run it with the `--stop-signal=SIGHUP` command option so that the shell is stopped cleanly when you stop the container. Many shells ignore the default SIGTERM signal when stopping a container. If the correct stop-signal isn't used, the container might return the following error when the container is stopped:

```
WARN[0010] StopSignal SIGTERM failed to stop container myol9 in 10 seconds,
resorting to SIGKILL
```

For more information on the `podman run` command, use the `podman-run(1)` manual page.

Example 4-2 Run a container and run a system command

```
podman run --rm oraclelinux:9-slim cat /etc/oracle-release
```

A container is started, and when the system command completes (the `cat` command in this example), the container is destroyed. The output looks similar to:

```
Oracle Linux Server release 9.6
```

Example 4-3 Run a container and connect to the Bash shell

```
podman run --name oracleshell -it oraclelinux:9-slim /bin/bash
```

A container is started, and the Bash shell is started. The output looks similar to:

```
bash-5.1#
```

You can run system command in the shell. For example:

```
cat /etc/oracle-release
```

The output looks similar to:

```
Oracle Linux Server release 9.6
```

The container stops as soon as you disconnect by typing `exit`.

To restart the container and connect to it again, run the `podmand start` command. For example:

```
podman start -ai oracleshell
```

Example 4-4 Running a container in the background

```
podman run -d --name oracledaemon oraclelinux:9-slim /bin/bash -c 'sleep 1000'
```

Example 4-5 Running a container with a shell as the primary process

```
podman run --stop-signal SIGHUP --name myol9 oraclelinux:9-slim
```

Enabling FIPS Mode in Containers

To run containers in FIPS mode, you must first enable FIPS mode on the Oracle Linux host system.

After you enable FIPS mode on an Oracle Linux host, Podman runs Oracle Linux containers in FIPS mode automatically.

For more information about enabling FIPS mode on Oracle Linux hosts, see the following documents:

- Oracle Linux 8: Enhancing System Security
- Oracle Linux 9: Enhancing System Security
- Oracle Linux 10: Enhancing System Security

> **Note**
>
> Oracle provides FIPS compliant container images by using the `slim-fips` tag. Container images tagged as FIPS compliant include compliant cryptographic package versions and initial image setup required for container FIPS mode. If you use these images you don't need to perform any extra steps to configure a container for FIPS mode. See The slim Tag for more information.

Listing and Monitoring Containers

Podman contains various commands to list and monitor containers. The examples here show you a few options.

Example 4-6 List the running containers

You can list all the running Podman containers using the `podman ps` command. Use the `--all` flag to also display the stopped and paused containers:

```
podman ps --all
```

The command shows similar output to the following:

```
CONTAINER ID  IMAGE
COMMAND               CREATED         STATUS                     PORTS
NAMES
dfb0dc74293a  container-registry.oracle.com/os/oraclelinux:9-slim  /bin/
bash              17 minutes ago  Exited (0) 12 minutes ago
oracleshell
1dd4aff270bb  container-registry.oracle.com/os/oraclelinux:9-slim  /bin/bash -
c slee...  10 minutes ago  Up 10 minutes                           oracledaemon
```

Example 4-7 Show container logs

To review the logs generated by a container that has already performed actions, use the `podman logs` command with the `NAME` or `CONTAINER ID` of the container. For example:

```
podman logs oracleshell
```

Example 4-8 Show container hardware resources

To review the hardware resource usage statistics for any running container, use the `podman stats` command. For example:

```
podman stats oracleshell
```

The command provides similar output to the following:

```
ID            NAME         CPU %       MEM USAGE / LIMIT  MEM %       NET
IO      BLOCK IO    PIDS        CPU TIME    AVG CPU %
bae740b48b53  oracleshell  0.00%       0B / 0B            0.00%       0B /
0B     0B / 0B     0           0s          0.00%
```

> **Note**
>
> To review statistics for containers without root permissions, you need to enable `cgroups` v2 on the host system. This isn't enabled by default for Oracle Linux 8 hosts, but is enabled default for Oracle Linux 9 and Oracle Linux 10 hosts. For more information about `cgroups` v2, see Run Control Group Version 2 on Oracle Linux.

Pausing and Resuming Containers

Pause and resume Podman containers using the `podman pause` and `podman unpause` commands.

If you need to temporarily halt the operation of a container without destroying its workload, use the `podman pause` command and specify the container name or ID.

Running the `podman pause` command freezes all the running processes inside a container, in their current state. When you're ready for the container to resume where it was halted, you can instruct the container to continue with its previous operation from that point by using the `podman unpause` command with the container name or ID.

For more information on the `podman pause` command, use the `podman-pause(1)` manual page. For information on the `podman unpause` command, use the `podman-unpause(1)` manual page.

Example 4-9 Pause a container

```
podman pause mycontainer
```

Example 4-10 Unpause a container

```
podman unpause mycontainer
```

Stopping and Removing Containers

Stop and remove Podman containers using the `podman stop` and `podman rm` commands.

To stop a container use the `podman stop` command with the name or container ID. If you need to temporarily take the server down for maintenance, you can stop every running container that hasn't already been paused by appending the `--all` flag to the `podman stop` command:

To delete a container use the `podman rm` command with the container name or ID. You can also use the `--all` flag to remove all containers.

For more information on the `podman stop` command, use the `podman-stop(1)` manual page. For information on the `podman rm` command, use the `podman-rm(1)` manual page.

Example 4-11 Stop a container

```
podman stop mycontainer
```

Example 4-12 Stop all containers

```
podman stop --all
```

Example 4-13 Remove a container

```
podman rm mycontainer
```

Example 4-14 Remove all containers

```
podman rm --all
```

5
Podman Pods

Podman introduces the concept of the "pod" within the context of a container runtime. This concept is borrowed from Kubernetes and isn't available in Docker.

A pod is a collection of containers that are grouped together into a single namespace so that they can share resources, such as local networking, to communicate with each other and interact. A pod can be used to group a set of services that you need to deploy a complete application.

In many ways a pod behaves in a similar way to a virtual host on which the services within each container are run. This means that each container can access the services on each other container as if they were running on the same host. Running containers in this way can remove a lot of complexity around networking and can make it easier to limit public exposure of ports that are only intended for use by services within the application itself.

Podman pods are the smallest compute units that can be created and deployed in a Kubernetes environment. These pods include an *infra* container so that Podman can connect with all the containers within the pod. Podman can manage the containers in the pod, such as stopping containers, without interfering with the operation of the pod itself.

By running containers within pods, it's more straightforward to set up and tear down entire application environments using atomic operations. By using pods, you can create service wrappers to automatically start a set of containers for an application at boot. See Podman Service Wrappers for more information.

Creating Pods

To create Podman pods, use the `podman pod create` command, or the `podman run` command with the `--pod` flag.

Create a pod with the `podman pod create` command. Include the `--name` flag to give the pod a human-readable identifier. You can also set the `--hostname` option if services within the pod need to use a particular hostname when connecting to each other.

Pods can be created automatically when a container is run for the first time. To do this, use the `podman run` command with the `--pod` option, and prepend the `new:` option to the name for the pod.

Attach containers to a pod using the `podman run` command with the `--pod` flag.

For more information on the `podman pod create` command, see the `podman-pod-create(1)` manual page.

Example 5-1 Create a pod with a name

Create a pod named `mypod`:

```
podman pod create --name mypod
```

Example 5-2 Create a container and an associated pod automatically

Create a pod named `mypod` that includes a container that runs an NGINX web server:

```
podman run --pod new:mypod --detach quay.io/libpod/alpine_nginx:latest
```

Example 5-3 Create and attach containers to a pod

1. Create a pod named `mypod`:

   ```
   podman pod create --name mypod
   ```

2. Create a container using an `nginx` image and connect it to the pod named `mypod`:

   ```
   podman run --pod mypod --detach quay.io/libpod/alpine_nginx:latest
   ```

3. Create a second container, using an `oraclelinux` image and connect it to the pod named `mypod`:

   ```
   podman run --pod mypod -it --rm oraclelinux:9-slim curl http://localhost:80
   ```

   The `curl` command is run in the second container to access the NGINX web service running on `localhost` on port 80 (running on the first container). The output from the `curl` command shows the HTML output of the NGINX server, and looks similar to:

   ```
   podman rulez
   ```

   The containers are both running as a standard user (not root), but can use a reserved port within the pod without any port mapping required. Furthermore, the containers can both use the `localhost` network namespace and can access each other as if they were running on the same host. This example provides an illustration of how pods can make it easier for services running within different containers to access each other and work together without any requirement for complex networking.

Listing and Monitoring Pods

Podman contains various commands to list and monitor pods. The examples here show you a few options.

Example 5-4 List pods

List all the available and running pods using the `podman pod ps` or `podman pod list` command. For example:

```
podman pod ps
```

Or:

```
podman pod list
```

Both commands show similar output to the following:

```
POD ID        NAME        STATUS      CREATED         INFRA ID      # OF
CONTAINERS
d2789902abe4  oraclepod   Created     9 seconds ago   4e203a8a2f6d  1
d8e8626a058c  mypod       Running     21 seconds ago  c31228fb0310  2
```

Example 5-5 List the containers in a pod

Review all the containers on the system using the `podman ps` command. Use the `--all` flag to show the containers, and the `--pod` flag to show the pods they're associated with.

```
podman ps --all --pod
```

You can also combine these flags using:

```
podman ps -ap
```

The output might look similar to:

```
CONTAINER ID  IMAGE
COMMAND               CREATED         STATUS         PORTS
NAMES               POD ID        PODNAME
7fefb402a3b8  localhost/podman-
pause:5.4.0-1750143108                                  45 seconds ago  Up 33
seconds                   9e746bbc3f6b-infra  9e746bbc3f6b  mypod
22061372871d  container-registry.oracle.com/olcne/nginx:1.20.1  nginx -g
daemon o...  32 seconds ago  Up 33 seconds  80/tcp, 443/tcp
awesome_driscoll    9e746bbc3f6b  mypod
```

In this example, the `mypod` pod is listed under `PODNAME` for each of the two containers, so these two containers are running in the same pod.

Example 5-6 List the containers in a named pod

Show the containers in pod named `mypod`:

```
podman pod ps --ctr-names --filter name=mypod
```

The output lists the containers in the pod, and might look similar to:

```
POD ID        NAME        STATUS      CREATED         INFRA ID      NAMES
8f7088f859da  mypod       Running     13 minutes ago  d2bb22029ec9
8f7088f859da-infra,nginxcontainer
```

Example 5-7 Inspect a pod

To show the configuration information about a pod, use the `podman pod inspect` command. For example:

```
podman pod inspect mypod
```

The JSON output looks similar to:


```
[
     {
          "Id":
"d8e8626a058c4bc538f95542728e6bdc8c44fb34b960ad9b81b06e4a804c4f3e",
          "Name": "mypod",
          "Created": "2025-07-11T03:26:07.8625531Z",
          "CreateCommand": [
               "podman",
               "run",
               "-d",
               "--pod",
               "new:mypod",
               "container-registry.oracle.com/olcne/nginx:1.20.1"
          ],
          "ExitPolicy": "continue",
          "State": "Running",
          "Hostname": "",
          "CreateCgroup": true,
          "CgroupParent": "user.slice",
          "CgroupPath": "user.slice/user-1000.slice/user@1000.service/
user.slice/user-
libpod_pod_d8e8626a058c4bc538f95542728e6bdc8c44fb34b960ad9b81b06e4a804c4f3e.sl
ice",
          "CreateInfra": true,
          "InfraContainerID":
"c31228fb03102c049f195f07da6415eee14267b55135ce480ef5a8f4d6660bbb",
          "InfraConfig": {
               "PortBindings": {},
               "HostNetwork": false,
               "StaticIP": "",
               "StaticMAC": "",
               "NoManageResolvConf": false,
               "DNSServer": null,
               "DNSSearch": null,
               "DNSOption": null,
               "NoManageHostname": false,
               "NoManageHosts": false,
               "HostAdd": null,
               "HostsFile": "",
               "Networks": null,
               "NetworkOptions": null,
               "pid_ns": "private",
               "userns": "host",
               "uts_ns": "private"
          },
          "SharedNamespaces": [
               "ipc",
               "net",
               "uts"
          ],
          "NumContainers": 2,
          "Containers": [
               {
                    "Id":
```

```
"c31228fb03102c049f195f07da6415eee14267b55135ce480ef5a8f4d6660bbb",
                    "Name": "d8e8626a058c-infra",
                    "State": "running"
               },
               {
                    "Id":
"971bc0b8146584f65f0cb71d93eade808540d9a956d2a2d6c6445d33adf0abea",
                    "Name": "eager_pascal",
                    "State": "running"
               }
          ],
          "LockNumber": 0
     }
]
```

Example 5-8 List processes running in a pod

To list the processes running in a pod, use the `podman pod top` command. For example:

```
podman pod top mypod
```

The output might look similar to:

```
USER        PID         PPID        %CPU        ELAPSED          TTY
TIME        COMMAND
0           1           0           0.000       15m43.454064014s  ?
0s          /catatonit -P
root        1           0           0.000       15m42.454997215s  ?
0s          nginx: master process nginx -g daemon off;
nginx       2           1           0.000       15m42.455054885s  ?
0s          nginx: worker process
nginx       3           1           0.000       15m42.455123415s  ?
0s          nginx: worker process
```

Example 5-9 Show hardware resource usage for pods

To show resource usage for containers in pods, use the `podman pod stats` command.

```
podman pod stats -a --no-stream
```

The output might look similar to:

```
POD           CID           NAME                CPU %       MEM USAGE/
LIMIT   MEM %       NET IO        BLOCK IO    PIDS
d8e8626a058c  c31228fb0310  d8e8626a058c-infra  0.00%       53.25kB /
16.29GB  0.00%       0B / 1.076kB  -- / --     1
d8e8626a058c  971bc0b81465  eager_pascal        0.00%       2.642MB /
16.29GB  0.02%       0B / 1.076kB  -- / --     3
d2789902abe4  4e203a8a2f6d  d2789902abe4-infra  0.00%       -- /
--            0.00%       -- / --       -- / --     --
```

To show real time resource usage, don't include the `--no-stream` flag.

```
podman pod stats -a
```

To exit the real time resource reporting, use `Ctrl+C`.

Pausing and Resuming Pods

Pause and resume Podman pods using the `podman pod pause` and `podman pod unpause` commands.

To temporarily halt the operation of a pod without destroying its workload, use the `podman pod pause` command and specify the pod name or ID.

Running the previous command freezes all the running processes inside a pod, in their current state. When you're ready for the pod to resume where it was halted, you can instruct the pod to continue with its previous operation from that point by using the `podman pod unpause` command with the pod name or ID.

For more information on the `podman pod pause` command, see the `podman-pod-pause(1)` manual page. For information on the `podman pod unpause` command, see the `podman-pod-unpause(1)` manual page.

Example 5-10 Pause a pod

```
podman pod pause mypod
```

Example 5-11 Unpause a pod

```
podman pod unpause mypod
```

Stopping and Starting Pods

Stop and start Podman pods using the `podman pod stop` and `podman pod start` commands.

Starting and stopping containers in a pod might affect the entire pod. However, you can use the `podman pod start` and `podman pod stop` commands to start and stop every container in a pod at the same time.

To stop a pod, use the `podman pod stop` command with the name or pod ID. If you need to temporarily take the server down for maintenance, you can stop every running pod by appending the `--all` flag to the `podman pod stop` command.

To start a pod, use the use the `podman pod start` command with the name or pod ID.

For more information on the `podman pod stop` command, see the `podman-pod-stop(1)` manual page. For information on the `podman pod start` command, see the `podman-pod-start(1)` manual page.

Example 5-12 Stop a pod

```
podman pod stop mypod
```

Example 5-13 Stop all running pods

```
podman pod stop --all
```

Example 5-14 Start a pod

```
podman pod start mypod
```

Removing Pods

Remove Podman pods using the `podman pod rm` command.

To delete a pod use the `podman pod rm` command with the pod name or ID. You can remove every running pod by appending the `--all` flag to the `podman pod rm` command.

Before you remove a pod it must be stopped. Use the `podman pod stop` command to stop pods.

Pods can only be removed when all the containers in the pod have been removed, except for the infra container. By default, an infra container is created for each pod, so a pod normally contains at least one container which can only be removed by removing the pod itself.

Example 5-15 Remove a pod

```
podman pod rm mypod
```

Example 5-16 Remove all running pods

```
podman pod rm --all
```

6
Kubernetes Files

Use the `podman kube` commands to generate and use YAML files that can be used in both Podman and Kubernetes to move pods, containers, volumes and other objects between the two container platforms.

Using Kubernetes YAML files provides several benefits, such as:

- **Portability**: You can move pods between Podman and Kubernetes, or run the same containers and pods on both container platforms.
- **Readability**: You can use the YAML files in programming languages.
- **Convenience**: The YAML files can contain all the necessary configuration information for the pods or containers. Thus, you don't need to specify different parameters when issuing Podman commands.

The following Podman commands are available to work with Kubernetes YAML files:

`podman kube generate`
Generates a Kubernetes YAML file of an existing pod. The YAML file contains a description of a Podman pod that can contain many containers, volumes, or other objects. You can use this YAML file to create pods in either Podman or Kubernetes.

`podman kube play`
Deploys a pod in Podman, based on a Kubernetes YAML file.

`podman kube apply`
Deploys a pod into a Kubernetes cluster, based on a Kubernetes YAML file.

`podman kube down`
Stops the containers in a pod, and removes the pod in Podman, using a Kubernetes YAML file.

To get more information about these commands, use the `--help` flag when issuing the commands. You can also see the `podman-kube(1)` manual page.

The examples in this section show you how to generate Kubernetes YAML files in Podman, and use the YAML files in both Podman and Kubernetes.

> **Tip**
>
> For Oracle Linux 8 and Oracle Linux 9 systems, the Oracle Cloud Native Environment `libvirt` provider can be useful for testing Kubernetes YAML files. For more information, see Oracle Cloud Native Environment Quick Start for Release 2.

Some examples in this section use the Kubernetes CLI (`kubectl`) to interact with a Kubernetes cluster. You can install the `kubectl` software package to the local system and use the Kubernetes configuration file (the `kubeconfig` file) to connect to the cluster. Information on installing `kubectl` is available in the Oracle Cloud Native Environment documentation or in the upstream Kubernetes documentation.

Podman has commands to generate a Kubernetes YAML file and deploy it to a Kubernetes cluster, but any further management of Kubernetes objects (such as pods and volumes) must be done using `kubectl` commands. For information on `kubectl` commands and how to use them, see the Kubernetes upstream documentation.

Example 6-1 Creating and using a Kubernetes YAML file

This example steps through creating a Podman pod, creating a Kubernetes YAML file from the pod, then deploying the pod to both Podman and Kubernetes using the YAML file.

1. Create a pod and add an NGINX container to the pod.

   ```
   podman run --pod new:mypod -dt --name nginxcontainer quay.io/libpod/
   alpine_nginx:latest
   ```

2. Verify the creation of the pod and the container.

   List all the pods:

   ```
   podman pod list
   ```

   The output looks similar to:

   ```
   POD ID        NAME        STATUS      CREATED        INFRA ID      # OF
   CONTAINERS
   8f7088f859da  mypod       Running     2 minutes ago  d2bb22029ec9  2
   ```

   Show the containers in the pod:

   ```
   podman pod ps --ctr-names --filter name=mypod
   ```

   The output lists the container names in the pod, and looks similar to:

   ```
   POD ID        NAME        STATUS      CREATED         INFRA ID      NAMES
   8f7088f859da  mypod       Running     13 minutes ago  d2bb22029ec9
   8f7088f859da-infra,nginxcontainer
   ```

   The infra and `nginxcontainer` are listed in the pod.

3. Create a Kubernetes YAML file from the pod.

   ```
   podman generate kube mypod --filename mypod.yaml
   ```

4. Check the contents of the YAML file. The `mypod.yaml` file might look similar to:

   ```
   # Save the output of this file and use kubectl create -f to import
   # it into Kubernetes.
   #
   # Created with podman-5.4.0
   apiVersion: v1
   kind: Pod
   metadata:
     annotations:
       io.kubernetes.cri-o.SandboxID/nginxcontainer: d2bb22029ec98f7...
     creationTimestamp: "2025-07-15T02:18:18Z"
   ```

```
  labels:
    app: mypod
  name: mypod
spec:
  containers:
  - env:
    - name: TERM
      value: xterm
    image: quay.io/libpod/alpine_nginx:latest
    name: nginxcontainer
    tty: true
```

5. Remove the containers and pod before trying to run this YAML file in Podman:

   ```
   podman stop nginxcontainer
   ```

   ```
   podman rm nginxcontainer
   ```

   ```
   podman pod rm mypod
   ```

6. Run the YAML in Podman to deploy the pod and its containers:

   ```
   podman kube play mypod.yaml
   ```

   You can see the pod is and the containers are running using the steps to verify the pod shown earlier. For example:

   ```
   podman pod list
   ```

   ```
   podman pod ps --ctr-names --filter name=mypod
   ```

7. Stop the Podman pod using the YAML file:

   ```
   podman kube down mypod.yaml
   ```

8. Run the YAML file in Kubernetes to deploy the pod.

   If you have the `kubeconfig` file for the Kubernetes cluster, you can use the `podman kube apply` command. For example:

   ```
   podman kube apply --kubeconfig mykubeconfig --file mypod.yaml
   ```

   If you have access to the Kubernetes cluster, you can copy the YAML file to a system that has access to the Kubernetes CLI, and use the `kubectl create` command to deploy the pod. For example:

   ```
   kubectl create -f mypod.yaml
   ```

9. To confirm the pod is running in Kubernetes, run the following command:

   ```
   kubectl get pods
   ```

10. To remove the Kubernetes pod, run the following command:

```
kubectl delete pod mypod
```

Creating a Kubernetes YAML File

Use the `podman kube generate` command to generate a Kubernetes YAML file for a Podman container, pod, or volume.

The YAML file can be used in a Kubernetes environment to start a Pod, Service, or PersistentVolumeClaim. It can also be used in Podman to test the Kubernetes objects are created correctly.

For pods that include volume mounts to start in Kubernetes, a StorageClass must be set up in Kubernetes with the appropriate configuration for the PersistentVolumeClaim. If the StorageClass is set up, a volume is automatically created when you run the pod.

For more information on the `podman kube generate` command, see the `podman-kube-generate(1)` manual page.

Example 6-2 Create a YAML file from a container

1. Create a Podman container:

```
podman create --name myoracle oraclelinux:9-slim
```

2. Generate a Kubernetes YAML file of the container:

```
podman generate kube myoracle --filename myoracle.yaml
```

3. Verify the resulting `myoracle.yaml` file looks similar to:

```
# Save the output of this file and use kubectl create -f to import
# it into Kubernetes.
#
# Created with podman-5.4.0
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2025-07-14T07:09:20Z"
  labels:
    app: myoracle-pod
  name: myoracle-pod
spec:
  containers:
  - image: container-registry.oracle.com/os/oraclelinux:9-slim
    name: myoracle
```

Example 6-3 Create a YAML file from a container with a volume mount

1. Create a Podman container that includes a volume mount:

```
podman run --name myvolumepod --volume mydata:/data oraclelinux:9-slim
```

2. Generate a Kubernetes YAML file of the container:

```
podman generate kube myvolumepod --filename myvolumepod.yaml
```

3. Verify the resulting `myvolumepod.yaml` file looks similar to:

```
# Save the output of this file and use kubectl create -f to import
# it into Kubernetes.
#
# Created with podman-5.4.0

# NOTE: If you generated this yaml from an unprivileged and rootless
podman container on an SELinux
# enabled system, check the podman generate kube manual page for steps to
follow to ensure that your pod/container
# has the right permissions to access the volumes added.
---
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2025-07-14T07:12:40Z"
  labels:
    app: myvolumepod-pod
  name: myvolumepod-pod
spec:
  containers:
  - image: container-registry.oracle.com/os/oraclelinux:9-slim
    name: myvolumepod
    volumeMounts:
    - mountPath: /data
      name: mydata-pvc
  volumes:
  - name: mydata-pvc
    persistentVolumeClaim:
      claimName: mydata
```

Example 6-4 Creating a YAML file from a pod

1. Create a pod and add an NGINX container to the pod.

```
podman run --pod new:mypod -dt --name nginxcontainer quay.io/libpod/
alpine_nginx:latest
```

2. Create a Kubernetes YAML file from the pod.

```
podman generate kube mypod --filename mypod.yaml
```

3. Verify the resulting `mypod.yaml` file looks similar to:

```
# Save the output of this file and use kubectl create -f to import
# it into Kubernetes.
#
# Created with podman-5.4.0
apiVersion: v1
kind: Pod
```

```
  metadata:
    annotations:
      io.kubernetes.cri-o.SandboxID/nginxcontainer: d2bb22029ec98f7...
    creationTimestamp: "2025-07-15T02:18:18Z"
    labels:
      app: mypod
    name: mypod
  spec:
    containers:
    - env:
      - name: TERM
        value: xterm
      image: quay.io/libpod/alpine_nginx:latest
      name: nginxcontainer
      tty: true
```

Running a Kubernetes YAML File

Use the `podman kube play` command to run a Kubernetes YAML file in Podman. This is useful to verify the file is generated correctly.

Use the `podman kube play` command to run a Kubernetes YAML file in Podman.

For more information on the `podman kube play` command, see the `podman-kube-play(1)` manual page.

Example 6-5 Run a Kubernetes YAML file in Podman

```
podman kube play mypod.yaml
```

You can verify the pod and containers are running using `podman pod` commands such as:

```
podman pod list
```

```
podman pod ps --ctr-names --filter name=mypod
```

Removing Pods Using a Kubernetes YAML File

Use the `podman kube down` command to stop and remove Podman pods using a Kubernetes YAML file.

The example here shows you how to stop and remove Podman pods that were created using a YAML file generated using the `podman kube generate` command. The `podman kube down` command is most likely to be used with the `podman kube play` command when testing Kubernetes YAML files. The `podman kube down` command stops and removes the pod and associated objects listed in the YAML file, such as secrets and volumes.

For more information on the `podman kube down` command, see the `podman-kube-down(1)` manual page.

Example 6-6 Stop and remove a pod using a YAML file

```
podman kube down mypod.yaml
```

Deploying to a Kubernetes Cluster

Use the `podman kube apply` command to run a Kubernetes YAML file and create pods and the associated objects in a Kubernetes cluster.

Use the `podman kube apply` command to run the YAML file generated in Podman.

The example here shows how to run a YAML file generated in Podman to create pods and containers in a Kubernetes cluster. This assumes you have set up a Kubernetes cluster. You need to provide the Kubernetes configuration file (the `kubeconfig` file) for the destination cluster, and this can be copied from the Kubernetes cluster to the local system. You also need to ensure the Kubernetes Server API port is open on the server running Kubernetes.

When you deploy a YAML file to a Kubernetes cluster, the pods are running on that cluster, not Podman, so any management of the Kubernetes objects needs to be done using `kubectl`. No Podman commands are available to manage objects in a Kubernetes cluster, other than as shown here, to deploy a YAML file.

For more information on the `podman kube apply` command, see the `podman-kube-apply(1)` manual page.

Example 6-7 Deploy a YAML file to a Kubernetes cluster

```
podman kube apply --kubeconfig mykubeconfig --file mypod.yaml
```

If you have access to the Kubernetes cluster, use the Kubernetes CLI (`kubectl`) to verify the pod is created using:

```
kubectl get pods
```

Deploying to a Kubernetes Cluster Using kubectl

Use the Kubernetes CLI (`kubectl`) to deploy a Kubernetes YAML file into a Kubernetes cluster.

The example here shows how to run a YAML file generated in Podman to create pods and associated objects in a Kubernetes cluster. This assumes you have set up a Kubernetes cluster, and the Kubernetes CLI (`kubectl`) is configured to access the cluster.

1. Copy the YAML file to the system where `kubectl` is set up to access the cluster.
2. Use the `kubectl create` command to run the YAML file generated in Podman.

   For example:

   ```
   kubectl create -f mypod.yaml
   ```

3. Verify the pod is created.

   ```
   kubectl get pods
   ```

7

Podman Storage

Set up and configure storage for Podman pods and containers, and for Buildah and Skopeo.

By default, container images are stored in `$HOME/.local/share/containers/storage/`. If you start a container using root privileges (using the `sudo` command), the images are stored in the `/var/lib/containers` directory. These locations conform to Open Container Initiative specifications. The separation of the local image repository for standard users ensures that containers and images maintain the correct permissions and that containers can run concurrently without affecting other users on the system.

All Podman related utilities take advantage of the same storage configuration. This means that Podman, Buildah, and Skopeo are all aware of the same storage locations for images available on the system.

These storage locations can be set up as mount points to take advantage of network storage, or dedicated local file systems, depending on requirements.

> ⚠ **Caution**
>
> When containers are run by a standard user (without the `sudo` command), Podman doesn't have the necessary permissions to access network shares and mounted volumes. If you intend to run containers as a standard user, only configure directory locations on the local file systems.

You can alter the configuration for Podman storage to address other requirements for a particular use case.

Setting Storage Configuration Options

System-wide Podman storage is configured in the `/etc/containers/storage.conf` file on Oracle Linux 8 and Oracle Linux 9 systems, and the `/usr/share/containers/storage.conf` file on Oracle Linux 10 systems.

You can override the system-wide storage configuration by creating a separate `$HOME/.config/containers/storage.conf` configuration file for any user. This file must be created as needed. For example, the following is an example of a storage configuration file for a user in the `$HOME/.config/containers/storage.conf` file:

```
[storage]
  driver = "overlay"
  runroot = "/run/user/1000"
  graphroot = "/home/oracle/.local/share/containers/storage"
  [storage.options]
    size = ""
    remap-uids = ""
    remap-gids = ""
    ignore_chown_errors = ""
```

```
remap-user = ""
remap-group = ""
mount_program = "/usr/bin/fuse-overlayfs"
mountopt = ""
[storage.options.thinpool]
  autoextend_percent = ""
  autoextend_threshold = ""
  basesize = ""
  blocksize = ""
  directlvm_device = ""
  directlvm_device_force = ""
  fs = ""
  log_level = ""
  min_free_space = ""
  mkfsarg = ""
  mountopt = ""
  use_deferred_deletion = ""
  use_deferred_removal = ""
  xfs_nospace_max_retries = ""
```

Configuration options are described in detail in the `containers-storage.conf(5)` manual page. The following descriptions describe the information in this example configuration file:

- `driver`: The storage driver is used to define how images and containers are stored. In Docker, there were options to use `overlay` or `overlay2` drivers, but Podman treats these as interchangeable to mean `overlay2`. Oracle has tested the `overlay2` driver with XFS, Ext4, and Btrfs where kernel support is available. Although you can change the storage driver to use another file system that's capable of layering, Oracle only supports the `overlay2` driver with the tested file systems.
- `graphroot`: The storage location where images are stored. As already mentioned, for standard users images are typically in `$HOME/.local/share/containers/storage/`. A legitimate use case to change this location, is where home directories might be NFS mounted.

  > **Important**
  >
  > If you change the `graphroot` location, you must ensure that SELinux labeling is correct for the new location.

  You can provide more storage locations for other image repositories by defining the paths to this in the `additionalimagestores` parameter within the `[storage.options]` section of the configuration file. Use this option to provide read-only access to shared images across networked storage.
- `runroot`: The default storage directory for all writable content for a container. The data within this directory is temporary and exists for the lifetime of the container. For a root user, the `runroot` location is often set to `/var/run/containers/storage`.

Depending on the storage driver that you use, different storage options might be available to use in the `[storage.options]` section of the configuration file. Some generic options that control user and group remapping are available to all drivers, and in all cases you can set a `size` parameter to apply quotas to container images.

Setting Up Container Mounts

Podman caters to automatically mounting particular directories on the host system into each container. This feature can be useful for sharing host secrets and authentication information with each container without storing the information within the images themselves. A common use case is that where system-wide private keys, certificates, or authentication credentials might be needed during a build process to provide access to external resources that a user might not have at their privilege level.

You can define other default mounts in `/usr/share/containers/mounts.conf` or in `/etc/containers/mounts.conf`. These entries are formatted as a colon-separated mapping between the source and destination directories, for example:

```
/src/dir/on/host:/run/target/on/container
```

See the `containers-mounts.conf(5)` manual page for more information.

You can also use the `--volume` or `-v` option when building an image or running a container from an image to mount a local directory into a container directory where required. For example:

```
sudo buildah bud -v /path/on/host:/path/on/container:rw -t newimage:1.0 .
```

```
sudo podman run --name mycontainer -d -v /path/on/host:/path/on/container:z
newimage:1.0
```

Other options are available for a volume mount. Notably the `-z` option helps where you might need to share an SELinux security context between several containers or between the container and the host system. Sharing an SELinux security context is useful when running a container as a non root user. This option is based on the SELinux multi level security (MLS) feature.

To restrict the volume to only the running container such that the volume's SELinux context isn't with other containers, use the `-Z` option. This option is based on the SELinux multi category security (MCS) feature.

See the `podman-run(1)` manual page for more information on `--volume` options. See the Oracle Linux: Administering SELinux for more information about SELinux contexts.

8
Podman Networking

Understand how to configure the different networking requirements for containers that run in Podman, or when working with images and temporary containers within Buildah.

Podman handles the networking of containers differently depending on whether the containers are run by the root or privileged user or by a standard user on the host system. The root user has considerably more power to change network infrastructure on the host, but the standard user has limited ability to alter network infrastructure.

In a network setup for Podman, containers that are running in a pod or a group share the same networking namespace, and therefore have access to the same IP and MAC addresses and port mappings. The shared namespace eases network communication between different containers, or between the host and the containers running on it. For more information about pods, see Podman Pods.

Unless indicated otherwise, all the networking procedures in this section can only be performed by the root or privileged user.

Setting a Proxy Server

Configure Podman to use proxy servers, both system wide, and for the Podman `systemd` service.

Podman automatically uses the system proxy settings for commands that you run and for any containers that you provision.

You can apply proxy settings on a system-wide basis by adding these to `/etc/profile`:

```
HTTP_PROXY=proxy_URL:port
HTTPS_PROXY=proxy_URL:port
```

Some services, such as the Cockpit web console, use the Podman API `systemd` service to interact with Podman.

> **Note**
>
> The Podman API service isn't required to use Podman. Only run this service if you use applications that use the API to interact with Podman, such as the Cockpit web console.

To set the system proxy environment variables for services that use the Podman API, you can create a `systemd` service drop-in.

1. Create the `/etc/systemd/system/podman.service.d` directory, if it doesn't already exist, to host `systemd` service drop-in configuration specific to the Podman API service.

   ```
   sudo mkdir -p /etc/systemd/system/podman.service.d
   ```

2. Create or edit the `/etc/systemd/system/podman.service.d/http-proxy.conf` file to contain contents similar to:

   ```
   [Service]
   Environment="HTTP_PROXY=proxy_URL:port"
   Environment="HTTPS_PROXY=proxy_URL:port"
   ```

   Replace *proxy_URL:port* with the URL and port number for the proxy server that you need to use.

3. Reload the `systemd` configuration changes and restart the Podman API service:

   ```
   sudo systemctl daemon-reload
   ```

   ```
   sudo systemctl restart podman
   ```

Configuring Port Mapping for Containers

Map networking ports to use with unprivileged Podman containers.

For unprivileged containers that are run by the standard user, without root permissions, Podman relies on port mapping to use the existing network infrastructure that's available on the host system. A standard user can't, and doesn't need to, configure specific network settings such as assigning IP addresses for containers. Podman handles the networking functionality for these containers automatically by performing port forwarding to container-based services.

In previous releases of Oracle Linux and Podman, `slirp4netns` provided a separate network configuration for each container, set its own gateway address, and provided Network Address Translation (NAT) for communication between unprivileged containers and the host when restricted port access was needed. Port publishing for an unprivileged user was limited to IPv4 port numbers 1024 to 65535, and there was no native IPv6 port mapping functionality.

For systems running Oracle Linux 9.5 or newer, and Podman 5.3 or newer, unprivileged containers use pasta networking by default. Oracle Linux 10 systems also use pasta networking by default.

Pasta networking uses the Plug A Simple Socket Transport (`passt`) network driver to copy the IP addresses from the host network adapter configuration and provide a translation layer between layer 2 network interfaces and layer 4 sockets for protocols such as TCP, UDP, and ICMP. This translation layer is accessible to unprivileged containers, provides access to IPv4 and IPv6 port mappings, and manages network access between containers on the same host. For more information, see upstream passt documentation.

> **Note**
>
> To learn more about Pasta and `slirp4netns` networking, see the Use Pasta Networking with Podman on Oracle Linux tutorial.

Example 8-1 Verify pasta networking in a container

To verify that a running container on an Oracle Linux 9 host system is using pasta networking, use the `podman inspect` command and check the `HostConfig.NetworkMode` setting. For example:

```
podman inspect --format='{{.HostConfig.NetworkMode}}' container_id
```

Replace *container_id* with the name or ID of the container. If the container is using pasta networking, the output looks similar to:

```
pasta
```

Example 8-2 Map a port on the host to a container

This example maps port 8080 on the host to the container port 80:

```
podman run --name mynginx -d -p 8080:80 quay.io/libpod/alpine_nginx:latest
```

The `-P` option can be used when running the container to enable Podman to automatically configure port mappings. However, the resulting configuration might be less predictable than you intend.

After a port mapping is established, that port can be accessed directly from the host where the container is running. In this example, the host can access port 80 on the container by opening a web browser to `http://localhost:8080`, or using a `curl` command. For example:

```
curl http://127.0.0.1:8080
```

In this example, the output looks similar to:

```
podman rulez
```

To view a container's port mappings directly, use the `podman port` command. For example:

```
podman port mynginx
```

If the container has active port mappings, the command output looks similar to:

```
80/tcp -> 0.0.0.0:8080
```

You can use the `podman port -a` command to view all port mappings for all the containers running on the host.

Example 8-3 Verify port access between containers

Because the containers and the host share the same network namespace, a container can communicate directly with another container by using the port mapping with the IP address or host name of the Oracle Linux host system.

Extending the previous example, run a second container with the following command:

```
podman run -it --rm oraclelinux:9-slim curl http://hostname:8080
```

Where *hostname* is the host name or IP address of the Oracle Linux host system.

To review inbound web traffic on the container running a web server, use the `podman logs` command:

```
podman logs container 2> /dev/null | grep "GET /"
```

Replace *container* with the container name or ID.

HTTP requests between Podman containers on the same host are logged with the host system gateway IP address rather than the public-facing IP address, because the pasta networking translation layer efficiently routes network traffic to the correct place.

If the host system has firewall software running, inbound traffic must be allowed on the mapped port for the container to be externally accessible to other host systems and remote clients. For example:

```
sudo firewall-cmd --add-port=8080/tcp --permanent
sudo firewall-cmd --reload
```

You can then connect to the container from a remote host using:

```
curl http://hostname:8080
```

Inspecting Container Networking

You can inspect the networking information for any container that you have created to obtain important information such as IP addressing, networks, or port mappings. Note that where a container is running under the root account, prefix the following commands with `sudo`, as appropriate.

To view IP addresses for a container, run:

```
podman inspect --format='{{.NetworkSettings.IPAddress}}' container
```

To view networks that are attached to a container, run:

```
podman inspect --format='{{.NetworkSettings.Networks}}' container
```

To view port mappings for a container, run:

```
podman inspect --format='{{.NetworkSettings.Ports}}' container
```

Advanced Networking for Containers

Advanced Podman network configuration can only be performed by the root user, and therefore applies only to containers that are run by the root user.

Advanced networking, which might require assigning IP addresses, enables containers to take advantage of particular features within the network stack to communicate with other containers in a pod. In this case, Podman implements a bridged network stack that can handle IP address assignment and full network access for each container.

For containers that are run by the root user, network management is achieved by using one of two backend network stacks:

- **Container Network Interface (CNI)**: A deprecated networking stack written in Golang and designed around the concept of plugins that can be used to implement different networking functions for a wide range of container related projects. For more information, see the upstream CNI documentation.
- **Netavark**: A network stack designed for Podman, but compatible with other Open Container Initiative container management applications. For more information, see the upstream Netavark documentation.

Podman selects which network stack to use automatically, depending on which network stacks are available on a system. You can identify which network stack a system is using by running:

```
podman info --format '{{.Host.NetworkBackend}}'
```

> **Caution**
>
> The `podman network` commands that are described in this section only work for containers that are run with root permissions. Running these commands with standard user containers returns an error code.

About CNI Networks

> **Note**
>
> The CNI network stack is now deprecated and might be removed in future releases of Podman. Consider using Netavark instead. To change network stack, see Changing the Network Backend. Although CNI is deprecated, Netavark doesn't support plugins available in CNI, such as the ability to connect to Kubernetes networks created using Flannel. You can continue to use CNI networking to take advantage of this facility, if required.

CNI is a set of network tooling that caters to container-based network requirements. CNI uses a plugin development model to accommodate different network functions and requirements. Podman can use many of these plugins directly to easily set up basic networking for individual containers, or for containers running within a pod.

You can opt to use CNI as the default network backend to maintain consistent configuration with older Podman deployments.

To use CNI, you must have the `containernetworking-plugins` package installed. To check whether this package is installed, use:

```
rpm -q containernetworking-plugins
```

If it's not installed, install it using:

```
sudo dnf install -y containernetworking-plugins
```

For each network you create in Podman, a new configuration file in JSON format is generated in the `/etc/cni/net.d/` directory. In most instances, you don't need to edit or manage the files in this directory.

A typical network configuration file might look similar to:

```
{
   "cniVersion": "0.4.0",
   "name": "mynetwork",
   "plugins": [
      {
         "type": "bridge",
         "bridge": "cni-podman1",
         "isGateway": true,
         "ipMasq": true,
         "hairpinMode": true,
         "ipam": {
            "type": "host-local",
            "routes": [
               {
                  "dst": "0.0.0.0/0"
               }
            ],
            "ranges": [
               [
                  {
                     "subnet": "10.89.0.0/24",
                     "gateway": "10.89.0.1"
                  }
               ]
            ]
         },
         "capabilities": {
            "ips": true
         }
      },
      {
         "type": "portmap",
         "capabilities": {
            "portMappings": true
         }
      },
      {
         "type": "firewall",
         "backend": ""
      },
      {
         "type": "tuning"
      }
   ]
}
```

About Netavark Networks

Netavark is a high-performance network stack that can be used to configure network bridges, firewall rules, and system settings for containers. Netavark doesn't use plugins to perform configuration setup. All network set up actions are performed directly by the tool itself, which reduces overhead, and improves network setup performance when you run a container. Netavark provides improved handling of IPv6, Network Address Translation (NAT) and port forwarding. DNS is also automatically configured across networks so that containers with several networks can connect to any other container on any other shared network by using the container name as a resolvable DNS reference.

Use the Netavark backend if all deployments are using Podman version 4.0 or later and you intend only to run containers within Podman. Netavark provides better performance and features that make containers easily integrate into existing network infrastructure and improved DNS resolution.

To use Netavark, you must have the `netavark` package installed. To check whether this package is installed, use:

```
rpm -q netavark
```

If it's not installed, install it using:

```
sudo dnf install -y netavark
```

For each network that you create within Podman, a new configuration file in JSON format is generated in the `/etc/containers/networks/` directory. In most instances, you don't need to edit or manage the files within these directories.

A typical network configuration file might appear as follows:

```
{
     "name": "mynetwork",
     "id": "3977b0c90383b8460b75547576dba6ebcf67e815f0ed0c4b614af5cb329ebb83",
     "driver": "bridge",
     "network_interface": "podman1",
     "created": "2022-09-06T12:08:12.853219229Z",
     "subnets": [
          {
               "subnet": "10.89.0.0/24",
               "gateway": "10.89.0.1"
          }
     ],
     "ipv6_enabled": false,
     "internal": false,
     "dns_enabled": true,
     "ipam_options": {
          "driver": "host-local"
     }
}
```

You can display the contents of a network configuration file with the following command:

```
sudo podman network inspect network_name
```

Changing the Network Backend

You can switch between using the CNI and Netavark network backend stacks and force Podman to select one over the other. Switching from one to the other assumes that you have the packages for both network backend stacks installed, namely `containernetworking-plugins` and `netavark`.

> **Important**
>
> If you change from one network backend to another, you must reset the Podman configuration. Switching network backends effectively removes all existing containers, images, networks, and pods from an environment.

To change and permanently set the network backend for a deployment, perform the following steps.

1. Check whether `/etc/containers/containers.conf` exists. If not, copy the default configuration to this location so that you can customize it for the deployment.

   ```
   sudo cp /usr/share/containers/containers.conf /etc/containers/
   ```

2. Edit `/etc/containers/containers.conf` and find the `network_backend` entry in the `[network]` section of the configuration file. Remove the entry's comment character if it exists. Change the value to match the network backend that you would prefer to use. For example, to use the CNI backend, change the entry to match:

   ```
   network_backend = "cni"
   ```

3. Reinitialize the Podman configuration to its pristine state:

   ```
   sudo podman system reset
   ```

   This command displays a warning and prompts you for confirmation.

   ```
   WARNING! This will remove:
           - all containers
           - all pods
           - all images
           - all networks
           - all build cache
           - all machines
   Are you sure you want to continue? [y/N] y
   ```

> **Note**
>
> If you have non root Podman instances, these also need to be reset individually if the network stack is changed.

4. Reboot the system to ensure that the networking is running correctly for Podman to function.

Creating and Removing Networks

Use the `podman network create` command to generate a new network configuration. Podman automatically defines network settings based on the default network and any other existing networks. However, options are available to set the network range, subnet size, and to enable IPv6. Use the `podman help network create` command and the `podman-network-create(1)` manual page to get more information about these options.

Use the `podman network remove` command to remove a network. You must first remove any containers connected to the network. You can use the `-f` option to force the removal of any containers that are using the network.

You can also remove all networks that are unused on the system using the `podman network prune` command.

Example 8-4 Create a network

```
podman network create mynetwork
```

Example 8-5 Remove a network

To remove a network that you have created, run:

```
sudo podman network rm mynetwork
```

Example 8-6 Remove unused networks

```
podman network prune
```

Listing Networks

Use the `podman network ls` command to print a list of all the Podman networks.

For more information on the `podman network ls` command, see the `podman-network-ls(1)` manual page.

Example 8-7 List all Podman networks

```
podman network ls
```

The output might look similar to:

```
NETWORK ID    NAME                         DRIVER
2f259bab93aa  podman                       bridge
0b723c8502a2  podman-default-kube-network  bridge
```

Connecting and Disconnecting Container Networks

Use the `podman network connect` command to add a container to a network. When you create containers, the network is automatically started and containers are assigned IP addresses within the range that's defined for a network. Likewise, when you delete a container, the network is also automatically stopped.

Use the `podman network disconnect` to remove a container from a network.

For more information on the `podman network connect` command, see the `podman-network-connect(1)` manual page. For information on the `podman network disconnect` command, see the `podman-network-disconnect(1)` manual page.

Example 8-8 Add a container to a network

```
podman network connect mynetwork mycontainer
```

Example 8-9 Disconnect a container from a network

```
podman network disconnect mynetwork mycontainer
```

9
Podman Quadlets

Create Podman Quadlets to easily generate and manage `systemd` services for containers, pods, and other Podman objects.

Instead of generating a Podman service wrapper by using the `podman generate systemd` command and then maintaining all the `systemd` unit files individually, you can maintain a single Podman Quadlet and rely on the `podman-systemd` daemon to regenerate those `systemd` unit files for you whenever they're needed.

The other practical benefit of using a Quadlet is that when a new version of Podman is released, you can roll out fixes and enhancements to any templated `systemd` unit files by reinitializing the affected services.

Quadlets are managed by using a unit file with and appropriate file extension (`.container`, `.volume`, `.network`, `.build`, `.pod`, or `.kube`). It must also include a Podman section (for example, `[Container]`). For example, a Quadlet for a container might include:

```
[Unit]
Description=your-description
After=local-fs.target

[Container]
Image=container_image
Exec=your-command-here

[Install]
# Start by default on boot
WantedBy=multi-user.target default.target
```

Creating Quadlets

Create Podman Quadlets that run containers, pods, volumes, and other Podman objects, using a `systemd` unit file, either for the root user, or for a user with standard permissions.

A system administrator can create Quadlets that run with root permissions. The Quadlet files use the file extensions `.container`, `.volume`, `.network`, `.build`, `.pod` and `.kube`. Save the Quadlet files in `/usr/share/containers/systemd/` or `/etc/containers/systemd/`.

> **Note**
>
> On Oracle Linux 8 hosts, Quadlets can only be run with root permissions. For more information, see Quadlets Fail For An Unprivileged User.

Quadlet files (with the file extensions of `.container`, `.volume`, `.network`, `.build`, `.pod` and `.kube`) can be created in the following locations, depending on the level of privilege for the Quadlet:

- A system administrator can create a Quadlet that runs with root permissions in:

  `/run/containers/systemd/` (temporary Quadlet location used for testing)

  `/etc/containers/systemd/` (system administrator defined Quadlets)

  `/usr/share/containers/systemd/` (distribution defined Quadlets)
- A system administrator can create rootless Quadlets that run when any user connects to the system in:

  `/etc/containers/systemd/users/`
- A system administrator can create rootless Quadlets that run when a user with a specific user ID connects to the system in:

  `/etc/containers/systemd/users/$USERID`
- A user can create Quadlets in:

  `$HOME/.config/containers/systemd/`

A list of all the options that can be included in a Quadlet file are available in the upstream Podman documentation, or the `podman-systemd.unit(5)` manual page.

The Quadlet files are used to generated `systemd` unit files (with a `.service` extension). Podman generates a unit file for each Quadlet file. The unit files can be managed like any other `systemd` service using the `systemctl` command. Each Quadlet file has a custom section for Podman objects (for example, `[Container]`) that's handled by Podman, and all other sections are standard `systemd` options and are passed on untouched. This means you can use standard `systemd` configuration in Quadlet file, including drop in files.

> **Tip**
>
> When a Quadlet starts a `systemd` service, any included containers are started. When a container is first started, Podman pulls the container images. To avoid any timeouts, it might be helpful to pull the container images before you start the `systemd` service created by a Quadlet. Or, you can increase the service startup timeout using the `TimeoutStartSec` option in the `[Service]` section of Quadlet files.

Quadlets require cgroup v2. To see if the system provides this, use:

```
podman info --format {{.Host.CgroupsVersion}}
```

> **Note**
>
> To learn how to enable cgroups v2 on Oracle Linux 8 hosts, see Run Control Group Version 2 on Oracle Linux.

Example 9-1 Create a Quadlet for a container

1. Create a unit file with the `.container` file extension. Save it to the appropriate location for the level of required user access. For example:

   ```
   [Unit]
   Description=My First Quadlet
   ```

```
[Container]
Image=container-registry.oracle.com/os/oraclelinux:9-slim
AutoUpdate=registry
Exec=sleep 60

[Service]
Restart=always
TimeoutStartSec=900

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

2. Reload the `systemd` services. To reload all `systemd` services, use:

   ```
   sudo systemctl daemon-reload
   ```

   To reload `systemd` services for a standard user on an Oracle Linux 9 or Oracle Linux 10 host, use:

   ```
   systemctl --user daemon-reload
   ```

3. Confirm the unit files are created. To examine the unit files that have been created, you can use the `-dryrun` option with the `quadlet` command:

   ```
   /usr/libexec/podman/quadlet -dryrun
   ```

   Add the `-user` option for Quadlet containers that run without root permissions. For example:

   ```
   /usr/libexec/podman/quadlet -dryrun -user
   ```

4. Start the `systemd` service. Manage the Quadlet the same way as any other `systemd` service using the `systemctl` command. For example, to start a Quadlet container called `myquadlet`, use:

   ```
   sudo systemctl start myquadlet.service
   ```

   For a service without root permission, include the `--user` option. For example:

   ```
   systemctl --user start myquadlet.service
   ```

5. Confirm the `systemd` service is running. To confirm the service is running, use the `systemctl status` command. For example:

   ```
   sudo systemctl status myquadlet.service
   ```

   For a service without root permission, include the `--user` option. For example:

   ```
   systemctl --user status myquadlet.service
   ```

6. Confirm the container is running.

To verify the container is running, use the `podman ps` command. For example, for a container running in a service that has root permissions, use:

```
sudo podman ps
```

Or, for a container running as a standard user:

```
podman ps
```

7. (Optional) Keep the service running after logout.

   If you want the `systemd` service to continue to run when a standard user logs out, enable lingering. For example:

   ```
   sudo loginctl enable-linger username
   ```

8. (Optional) Remove the service.

   To clean up after trying this example, remove the service and container using:

   ```
   systemctl stop myquadlet.service
   ```

   Or for a standard user:

   ```
   systemctl --user stop myquadlet.service
   ```

   The `systemd` service is stopped, which also stops the container.

Creating Quadlets With the Podman Shell

On Oracle Linux 9 and Oracle Linux 10 hosts, you can use the Podman login shell to run shell commands in a container. This feature provides the advantage of ensuring that users are kept within established boundaries while working inside containers. Users are configured to use the `/usr/bin/podmansh` shell instead of normal shells such as `/bin/bash`. The command starts the user's session in a rootless container called `podmansh`. This container continues to run until the user exits the session, at which point the container is removed.

Administrators configure users to use this feature through Quadlet files. These files define the parameters for users when they work in containers, such as the visibility of the host system when users are in the containers, access limitations, security privileges, and resource usage. For more information, see Podman Quadlets.

All the settings defined for the user in the Quadlet file transparently run without any need for user or administrator intervention.

The Podman login shell and other defined parameters combine to isolate the user and therefore ensures container security. The container is removed at the end of a session, and logging back in re-creates the same confined environment for the user. So, working with containers becomes more straightforward.

The following is an example of how you can configure a user to use the Podman login shell and other set parameters when working in a container. Assume that the Quadlet applies to a specific `$USERID`, and the Quadlet file is named `podmansh.container`.

```
[Unit]
Description=The Podmansh container
After=local-fs.target

[Container]
Image=container-registry.oracle.com/os/oraclelinux:9-slim
ContainerName=podmansh
RemapUsers=keep-id
RunInit=yes

Exec=sleep infinity

[Install]
RequiredBy=default.target
```

A list of all the options that can be included in this file are available in the upstream Podman documentation, or the `podman-systemd.unit(5)` manual page.

> **Note**
>
> Note that the name of the container (`ContainerName`) must be `podmansh`. This sets the `/usr/bin/podmansh` shell to run `podman exec` in the container.

The user with the ID that matches the Quadlet's `$USERID` would log in as follows:

```
ssh user@systemname
```

More information about using the `podmansh` shell is available in the upstream Podman documentation, or the `podmansh(1)` manual page.

Quadlet Services

All `systemd` services created by Quadlets are managed using the `systemctl` command.

After you have created a `systemd` service for any containers, pods, or other Podman objects, you can use `systemctl` commands to manage those services, and therefore all the Podman objects in the services. In the same way you manage regular `systemd` services, you can start, restart, stop and check the service status using `systemctl`. Enabling services to boot automatically is a little different as services created using Quadlets are transient services. If you're running containers as a standard user on an Oracle Linux 9 or Oracle Linux 10 host, all `systemctl` commands must include the `--user` option.

> **Tip**
>
> To try out using `systemd` services, see the Use Systemd on Oracle Linux tutorial.

Starting Services

To start a `systemd` service created by a Quadlet, use the `systemctl start` command.

Example 9-2 Start a Quadlet service

```
sudo systemctl start myquadlet.service
```

As a standard user, include the `--user` option. For example:

```
systemctl --user start myquadlet.service
```

Restarting Services

Restart the service using the `systemctl restart` command. Restarting the service stops and re-creates all objects in the service (containers, pods, and so on).

Example 9-3 Restart a Quadlet service

```
sudo systemctl restart myquadlet.service
```

For a standard user, include the `--user` option. For example.

```
systemctl --user restart myquadlet.service
```

Stopping Services

Stop a `systemd` service using the `systemctl stop` command. Stopping a service created by a Quadlet stops all objects in the service (containers, pods, and so on).

Example 9-4 Stop a Quadlet service

```
sudo systemctl stop myquadlet.service
```

To stop the service as a standard user, include the `--user` option. For example:

```
systemctl --user stop myquadlet.service
```

Checking the Service Status

Check the current status of a `systemd` service with the `systemctl status` command.

Example 9-5 Check the status of a Quadlet service

```
sudo systemctl status myquadlet.service
```

To see the service status as a standard user, include the `--user` option. For example:

```
systemctl --user status myquadlet.service
```

Enabling Services

Services created by Quadlets are considered transient services, and they don't have the same persistence as regular `systemd` services. Because of this, you can't use the `systemctl enable` command to set them to start when the host OS boots. Instead, the Podman generator uses the information in the `[Install]` section of Quadlet files when generating service unit files. For example, you can set the service to start when the OS boots by including:

```
[Install]
WantedBy=default.target
```

If services are running as a standard user, you can enable lingering so services start automatically when the user signs in to the OS. For example:

```
sudo loginctl enable-linger user
```

10

Podman Service Wrappers

Podman can integrate with `systemd` services to manage pods and containers as system services. By using Podman service wrappers, you can configure containers or pods to start at system boot and you can manage them similarly as other services that run on the host system.

> ⚠ **Caution**
>
> The method of generating `systemd` unit files in this section was deprecated in Podman 4.6. Instead, use Quadlets to create `systemd` services. For more information, see Podman Quadlets.

Podman provides the tools to automatically generate `systemd` service wrapper configuration files for any containers or pods on the system, so that you can manage container infrastructure using `systemd`. You can use the `podman generate systemd` command to automatically generate `systemd` unit files.

You can use `systemd` user services if you're running containers as a standard user, or you can configure system level services if you're running containers as the root user.

Generating Podman Service Wrappers

Use the `podman generate systemd` command to automatically generate `systemd` unit files for Podman containers and pods.

Instead of writing a `systemd` service wrapper from scratch, you can use the `podman generate systemd` command to automatically generate the service configuration file.

If you intend to run containers as root user system services, store the container service wrapper configuration files in `/etc/systemd/system/`. If you intend to run containers as a standard user, save the container service wrapper configuration files in `$HOME/.config/systemd/user/`.

Example 10-1 Generate a systemd service wrapper for a container

To generate a `systemd` service wrapper for an individual container, and store it in the `$HOME/.config/systemd/user` directory:

```
podman generate systemd --name containername > $HOME/.config/systemd/user/
container-containername.service
```

Example 10-2 Generate a systemd service wrapper for a pod

To generate a Podman service wrapper for a specific pod, use the following command:

```
podman generate systemd --name podname
```

However, to include generating service wrapper configuration files for all the containers within a pod itself, use the `--file` option with the command. In this case, run the command in the directory where you intend to generate the files.

Suppose that in `$HOME/.config/systemd/user`, you want to generate Podman service wrappers for both `mypod` and its containers. You would run the following commands:

```
cd $HOME/.config/systemd/user/
podman generate systemd --files --name mypod
```

With this command, the service wrapper that's responsible for `mypod` includes dependencies on each of the container wrappers that are required for the pod to run successfully.

If you start or stop the pod by using its `systemd` service wrapper, the container services automatically trigger the same action.

Starting and Restarting Podman Services

> ⚠ **Caution**
>
> If a container or pod is already running outside of the `systemd` service wrapper, the service wrapper is unable to start the container or pod. If so, use the `podman stop` or `podman pod stop` command to stop the container or pod first.

As a root user, you can start a container if its service configuration is stored in `/etc/systemd/system/`, for example:

```
sudo systemctl start container-containername.service
```

As a standard user, if you stored a service configuration in `$HOME/.config/systemd/user`, you can start the container in the same way but you must use the `--user` option:

```
systemctl --user start container-containername.service
```

Starting the service wrapper for a pod uses a parallel command syntax, as follows:

```
sudo systemctl start pod-podname.service
```

You can restart the service wrapper for a container or pod by using the `systemctl restart` command. The following command restarts a pod as a standard user:

```
systemctl --user restart pod-podname.service
```

If you start or restart a pod, all containers that are part of the pod are equally started or restarted.

Stopping Podman Services

You can stop a container or pod by using the `systemctl stop` command. The following command stops a pod as a standard user:

```
systemctl --user stop pod-podname.service
```

If you start or restart a pod, all containers that are part of the pod are equally started or restarted.

Checking the Status of Podman Services

You can check the current status of any service wrapper you create for containers or pods with the `systemctl status` command, for example:

```
systemctl --user status container-containername.service
```

Enabling Automated Restore for Podman Services

You can add custom configuration steps when you generate service wrappers for Podman containers.

For example, to create a service wrapper that always restarts after a one second timeout, set the `--restart-policy` flag with a parameter value, as shown:

```
sudo systemctl generate systemd --restart-policy=always -t 1 containername
> /etc/systemd/user/container-containername.service
```

To set the service wrapper to run automatically when the system starts up, type:

```
sudo systemctl enable container-containername.service
```

You can use the same commands with the service wrapper for a pod:

```
sudo systemctl enable pod-podname.service
```

If services are running as a standard user, you would need to give the user permission to run processes when they're not logged in. Otherwise, the user can't enable the service. Type the following command as the root user:

```
sudo loginctl enable-linger user
```

Tip

To try out using `systemd` services, see the Use Systemd on Oracle Linux tutorial.

Changing Podman Service Wrapper Configuration

The `systemd` service wrapper configuration files that are generated by Podman follow standard `systemd` configuration format and specification. You can change any of the service wrapper configuration files that are generated by manually editing these files within a text editor.

Change the behavior of `systemd` services wrappers on Oracle Linux by following the instructions in these books:

- Oracle Linux 8: Managing the System With systemd
- Oracle Linux 9: Managing the System With systemd
- Oracle Linux 10: System Management with systemd

For more information about how you can make modifications to the service wrapper you have generated with the `podman generate systemd` command, see the upstream Podman documentation.

Setting SELinux Permissions for Service Wrappers

If you have set SELinux to `enforcing` mode on the system, you must turn on the `container_manage_cgroup` permission so that `systemd` can be used to start, stop, and monitor containers:

```
sudo setsebool -P container_manage_cgroup on
```

11 Buildah

Use the Buildah utility to create custom Open Container Initiative compliant container images to use with Podman.

The Buildah utility is functionally similar to Podman in the way that it behaves, but maintains independence from Podman to build Open Container Initiative compliant images. The primary difference between Buildah and Podman is in the way the `run` command is handled. Because the purpose of Buildah is to build images, the `run` command behaves the same as a `RUN` statement within a Containerfile, which is a configuration file that contains the settings to automate the creation of a container image. This difference makes it easy to separate image builds from production level container infrastructure, running in Podman, and to easily process existing Containerfile build instructions.

Use Buildah to pull images from existing registries and to change them to create new images with more specialized functionality. You can use Buildah with an existing Containerfile to create an image, or alternately you can pull an image directly and change it within a container running within the Buildah environment.

Podman and Buildah use the same local image store, which means you can start containers in Podman from images that you have built in Buildah. You can also use images that have been pulled locally by Podman as the base images which you use to build new images using Buildah. While local images are shared, the containers themselves run separately within Buildah and Podman. Podman is unable to access containers running within Buildah and Buildah is unable to access containers running within Podman. This is because the containers that run in Buildah are used precisely to run commands to build a new image.

It might be useful to create a private container registry to store the images you build with Buildah. A private registry can be used to push the images you create, and pull from other Podman clients that might need to use the images to build containers. For information on creating a registry, see Private Container Registries.

For more information about the Containerfile format and build instructions see the `containerfile(5)` manual page.

For a complete listing of Buildah commands, use the `buildah --help` command, or view the `buildah(1)` manual page.

> **Note**
>
> Most Buildah operations don't require the use of `sudo`. We recommend you use Buildah as a standard user. Where privileged access is required, use the `buildah unshare` command to enter into a privileged Buildah shell. For more information, see the `buildah-unshare(1)` manual pages.

Creating an Image From a Containerfile

Build a container image from a Containerfile using the `buildah build` command.

Buildah is designed to work directly with an existing Containerfile and processes the file to build a container image using the `buildah build` command (which is an alias for the `buildah build-using-dockerfile` and `buildah bud` commands). The `buildah build` command behaves similarly to the `docker build` command.

You can use the default filenames of `Containerfile` or `Dockerfile` to specify container images. If you use a different filename, or a name with a file extension, these aren't recognized unless you include the `-f` option with the `buildah build` command.

To find out more about using the `buildah commit` command, see the `buildah-commit(1)` manual page.

1. Create a Containerfile.

   Find or create a Containerfile that specifies the contents of the container image. For example, create a file named `Containerfile` that contains:

   ```
   # This image is based on the latest Oracle Linux 9 image
   FROM container-registry.oracle.com/os/oraclelinux:9

   # Install OS updates and the httpd software package
   RUN echo "Updating all Oracle packages"; dnf -y update; dnf -y clean all
   RUN echo "Installing the httpd software package"; dnf -y install httpd &&
   dnf -y clean all

   # Expose the default httpd server port 80
   EXPOSE 80

   # Run the httpd server
   CMD ["/usr/sbin/httpd", "-DFOREGROUND"]
   ```

2. Build the image from the Containerfile.

   Use the `buildah build` command to build the Containerfile. Include the `--tag` option to tag the image. Include the `--file` option to specify the Container filename if the file doesn't use the default naming, or includes a file extension. For example:

   ```
   buildah build --tag myimage .
   ```

   The image is built, tagged, and added to the local image list.

3. Validate the image.

   To confirm that the new image is available, use the `buildah images` command. For example:

   ```
   buildah images
   ```

In this example, the image is in the local Podman storage (used by both Buildah and Podman), and the output looks similar to:

```
REPOSITORY                      TAG                 IMAGE ID
CREATED          SIZE
localhost/myimage               latest              761c8f4b4f98   6
minutes ago    279 MB
```

4. Verify the image can be started in a container.

   The `buildah from` command can be used to start a container that uses the image. For example:

   ```
   buildah from myimage
   ```

5. Verify the container is started.

   Use the `buildah containers` command to show the container is running. For example:

   ```
   buildah containers
   ```

   The output looks similar to:

   ```
   CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME
   CONTAINER NAME
   076b44920a9a     *     761c8f4b4f98 localhost/myimage:latest
   myimage-working-container
   ```

Creating a Buildah Container from an Image

Pull an image from a container registry and use it to create a Buildah container.

Buildah can pull images from a container registry using the `buildah pull` command, in the same way that you would pull an image using Podman. Start a Buildah container from an image with the `buildah from` command. You can run these commands separately, or combine them.

1. Pull an image.

   Use the `buildah pull` command to pull an image from a container registry. For example:

   ```
   buildah pull container-registry.oracle.com/os/oraclelinux:9-slim
   ```

2. Start a Buildah container from the image.

   Use the `buildah from` command to start a Buildah container from an image. For example:

   ```
   buildah from oraclelinux:9-slim
   ```

> **Tip**
>
> You can combine the `buildah pull` command with the `buildah from` command to start a container from an image hosted on a remote registry. For example:
>
> ```
> buildah from container-registry.oracle.com/os/oraclelinux:9-slim
> ```

3. Verify the container is started.

   Use the `buildah containers` command to show the container is running. For example:

   ```
   buildah containers
   ```

   The output looks similar to:

   ```
   CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME
   CONTAINER NAME
   ea27cc0312ba     *     bc1c39bd670d localhost:5000/ol9image:v1
   oraclelinux-working-container
   ```

Changing an Image

Interact with a Buildah container to install software, add files, and mount the container's volume.

You change images by running them as container instances within Buildah and using the `buildah run` command. This command functions in the same way as `RUN` statements in a `Containerfile`. For example, you can use the `buildah run` command to run shell commands in the Buildah container, such as installing packages and making OS changes.

You can also use other commands such as `buildah copy` and `buildah mount` to interact with Buildah containers.

Many of the examples here assume a Buildah container is already created and running that includes an Oracle Linux 9 image. The command to create this container is:

```
container=$(buildah from container-registry.oracle.com/os/oraclelinux:9)
```

The Buildah container name is set as a `$container` variable to make it easier to try the examples.

Example 11-1 Install a package to the Buildah container

Use the `buildah run` command to install a software package on a container. This example installs `nginx`.

```
buildah run $container dnf -y install nginx
```

Example 11-2 Run a shell command in a Buildah container

Use the `buildah run` command to display the Oracle Linux release in a Buildah container.

```
buildah run $container cat /etc/oracle-release
```

The output looks similar to:

```
Oracle Linux Server release 9.6
```

Example 11-3 Copy a file to a Buildah container

Use the `buildah copy` command to a copy file to a Buildah container.

Create a file named `mytext.txt` with some content. For example:

```
This is some text.
```

Copy the `mytext.txt` file to the `/tmp` directory on the container.

```
buildah copy $container mytext.txt /tmp
```

Use the `buildah run` command to show the contents of the file using the `cat` command.

```
buildah run $container cat /tmp/mytext.txt
```

The output looks similar to:

```
This is some text.
```

Example 11-4 Mounting a Buildah container's volume

You can make file changes inside the working container by mounting its root file system on the local host:

First, you need to enter into the Buildah namespace that has elevated permissions. This uses the overlay driver in rootless mode, and is known as a Buildah *unshare* session. For more information, see the `buildah-unshare(1)` manual pages.

```
buildah unshare
```

The command line user changes to show the user is now `root`.

Create a container in the unshare session and mount the container's volume.

```
container=$(buildah from oraclelinux:9)
mnt=$(buildah mount $container)
```

You can now interact with the volume mount. For example, install a software package on the container's volume:

```
dnf install --installroot $mnt httpd
```

When finished, unmount the container.

```
buildah unmount $container
```

Return to the original shell using the `exit` command.

Committing a Buildah Container to an Image

After completing the changes to a container, use the `buildah commit` command to generate a new image based on the current status of the Buildah container.

Changes to a Buildah container are temporary until they're committed to an image. You can commit a Buildah image to the local storage so it's immediately available in Podman. Buildah also provides the option to push the image to a container registry, and has options to authenticate with a registry, such as user authentication and certificate validation. You must have write access to a registry, so it might be useful to create a private registry for this purpose. For information on creating and using a private registry, see Private Container Registries.

To permanently store the changes to a Buildah container in an image, use the `buildah commit` command. To find out more about using the `buildah commit` command, see the `buildah-commit(1)` manual page.

Example 11-5 Commit a Buildah image to the local storage

Commit a Buildah container to an image the local storage. Include the `--rm` option to remove the working container from the Buildah environment at the same time.

```
buildah commit --rm oraclelinux-working-container myol9:v1
```

You can confirm the image is created using:

```
buildah images
```

The output looks similar to:

```
REPOSITORY                    TAG               IMAGE ID
CREATED         SIZE
localhost/myol9               v1                6afd745ed1cf   8 seconds
ago   519 MB
```

Example 11-6 Commit a Buildah image to a private secure registry

Commit a Buildah container and push the image to a private secure registry.

```
buildah commit --rm oraclelinux-working-container docker://
myregistry.example.com:5000/myol9:v1
```

Committing directly to a registry is the same as performing a local commit and then pushing it to a registry later using the `buildah push` command.

Verify the image can be pulled using Buildah:

```
buildah pull myregistry.example.com:5000/myol9:v1
```

Example 11-7 Commit a Buildah image to a private insecure registry

Commit a Buildah container and push the image to a local insecure registry.

```
buildah commit --tls-verify=false --rm oraclelinux-working-container docker://
localhost:5000/myol9:v1
```

Verify the image can be pulled using Buildah:

```
buildah pull --tls-verify=false localhost:5000/myol9:v1
```

Pushing an Image to a Registry

Use the `buildah push` command to push container images from the local storage to a container registry.

Buildah and Podman handle images interchangeably, so most of the commands that you use to work with images are replicated across these tools and perform the same operation. For example, the `buildah push` and `podman push` commands behave identically. You can use either of these commands to push an image to a container registry where you have write access. You might want to create a private registry to push and share images if you don't already have write access to a registry. For information on creating a registry, see Private Container Registries.

Buildah and Podman can also push images to other formats. This means you can create archive formats that you can share and reuse if you don't have access to a registry. For example, you can create a Docker compatible archive to create an archive file that's similar to using the `docker save` command. Use the format:

```
buildah push imagename docker-archive:/path/to/archive-file:image:tag
```

Change `docker-archive` in the command to `oci-archive` to generate an archive that complies with the Open Container Initiative specification.

For more information on the `buildah push` command, see the `buildah-push(1)` manual page.

Example 11-8 Push an image to a secure private registry

```
buildah push ol9image:v1 docker://myregistry.example.com:5000/myol9:v1
```

Verify the image can be pulled using Buildah:

```
buildah pull myregistry.example.com:5000/myol9:v1
```

Example 11-9 Commit an image to a private insecure registry

```
buildah push --tls-verify=false ol9image:v1 docker://localhost:5000/myol9:v1
```

Verify the image can be pulled using Buildah:

```
buildah pull --tls-verify=false localhost:5000/myol9:v1
```

Example 11-10 Create a Docker compatible archive file

```
buildah push ol9image:v1 docker-archive:/tmp/ol9image.tar:myol9:v1
```

An archive file is created in `/tmp/ol9image.tar` that contains the image.

Removing a Buildah Container

Remove Buildah containers using the `buildah rm` command. This command doesn't affect Podman containers.

The `buildah rm` command stops and removes a container from the Buildah container list. It removes the working container and unmounts any volume mounts that might have been set up for it. After the container is removed, any changes you made to the working container are lost and can't be retrieved, unless you created an image of the container before you removed it.

The `buildah rm` only removes the working container from the Buildah environment. Any containers running under Podman, or any existing images are unaffected.

To find out more about using the `buildah rm` command, see the `buildah-rm(1)` manual page.

You can also remove containers as you commit them to an image, by including the `--rm` option with the `buildah commit` command to both commit a working container and remove it from the Buildah container list at the same time. See Committing a Buildah Container to an Image.

Example 11-11 Remove a Buildah container

```
buildah rm oraclelinux-working-container
```

Example 11-12 Remove all Buildah containers

```
buildah rm --all
```

Removing an Image

Remove images using the `buildah rmi` command. This also removes images in Podman.

The `buildah rmi` command removes a images from the Buildah and Podman image list. Because images are shared between Podman and Buildah, removing an image with this command also removes it for Podman.

To find out more about using the `buildah rmi` command, see the `buildah-rmi(1)` manual page.

Example 11-13 Remove an image

```
buildah rmi localhost/myimage:latest
```

Example 11-14 Remove all images

```
buildah rmi --all
```

Inspecting an Image or Container

The `buildah inspect` command prints the metadata and history of an image or container created with Buildah. The output is a JSON array and might be helpful to identify historical changes and other information about the image or container.

To find out more about using the `buildah inspect` command, see the `buildah-inspect(1)` manual page.

Example 11-15 Inspect a container

Use the `buildah inspect` command to inspect a container.

```
buildah inspect image --type container oraclelinux-working-container
```

Example 11-16 inspect an image

Use the `buildah inspect` command to inspect an image.

```
buildah inspect --type image myol9:v1
```

12
Skopeo

Use Skopeo to inspect and copy Open Container Initiative compliant images between container storage types.

Skopeo is an optional utility that you can install in addition to Podman to inspect images in remote registries, and copy images between different types of Open Container Initiative compatible container storage. Skopeo doesn't require a running daemon to function.

> **Note**
>
> You don't need root permissions to run Skopeo commands. If any errors are returned, ensure that you have configured the appropriate proxy server settings, and that you have the necessary access permissions for the remote registries that you're using.
>
> For more information, see the `skopeo(1)` manual page.

Inspecting an Image

Use the `skopeo inspect` command to inspect information about an image in a container registry.

The `skopeo inspect` command prints information about a container, such as when it was created, its SHA digest signature, and the environment variables that are set for the image. The information can also be useful for inspecting the available tags for an image name in a registry.

For more information about the `skopeo inspect` command, see the `skopeo-inspect(1)` manual page.

Example 12-1 Inspect an image in a container registry

```
skopeo inspect docker://container-registry.oracle.com/os/oraclelinux:9
```

The output looks similar to:

```
{
    "Name": "container-registry.oracle.com/os/oraclelinux",
    "Digest":
"sha256:afeedad1979892ba77973ea9900dc2604873651b0ee8868bffff8c7987e32bb3",
    "RepoTags": [
        "10-slim",
        "10",
        "5.11",
        "5",
        "6-slim",
        "6.10",
        "6.6",
```

```
            "6.7",
            "6.8",
            "6.9",
            "6",
            "7-slim-amd64",
            "7-slim-arm64v8",
            "7-slim-fips-amd64",
            "7-slim-fips-arm64v8",
            "7-slim-fips",
            "7-slim",
            "7.0",
            "7.1",
            "7.2",
            "7.3",
            "7.4",
            "7.5",
            "7.6",
            "7.7",
            "7.8",
            "7.9",
            "7",
            "8-arm64v8",
            "8-slim-arm64v8",
            "8-slim-fips-amd64",
            "8-slim-fips-arm64v8",
            "8-slim-fips",
            "8-slim",
            "8.0",
            "8.1",
            "8.10-slim-fips",
            "8.10-slim",
            "8.10",
            "8.2",
            "8.3",
            "8.4",
            "8.5",
            "8.6",
            "8.7",
            "8.8",
            "8.9-slim-fips",
            "8.9-slim",
            "8.9",
            "8",
            "9-slim-fips",
            "9-slim",
            "9",
            "9test",
            "latest"
        ],
        "Created": "2025-08-01T17:33:19.437546569Z",
        "DockerVersion": "",
        "Labels": {
            "io.buildah.version": "1.33.11"
        },
        "Architecture": "amd64",
        "Os": "linux",
```

```
    "Layers": [

"sha256:25ef70384958aefe7777e15722705fbaccd630a1aabe14030e2ad22b3c205c37"
    ],
    "LayersData": [
        {
            "MIMEType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
            "Digest":
"sha256:25ef70384958aefe7777e15722705fbaccd630a1aabe14030e2ad22b3c205c37",
            "Size": 99740095,
            "Annotations": null
        }
    ],
    "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ]
}
```

Example 12-2 Inspect the default configuration for an image in a container registry

Review the default configuration settings and build history of an image by including the `--config` option.

```
skopeo inspect --config docker://container-registry.oracle.com/os/
oraclelinux:9
```

The output looks similar to:

```
{
    "created": "2025-08-01T17:33:19.437546569Z",
    "architecture": "amd64",
    "os": "linux",
    "config": {
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin"
        ],
        "Cmd": [
            "/bin/bash"
        ],
        "Labels": {
            "io.buildah.version": "1.33.11"
        }
    },
    "rootfs": {
        "type": "layers",
        "diff_ids": [

"sha256:0ae592ddaa5b488f567f350e0367e2b3f91ea04a23ea8931017916ecc0f922e4"
        ]
    },
    "history": [
        {
            "created": "2025-08-01T17:33:17.828674963Z",
            "created_by": "/bin/sh -c #(nop) ADD
```

```
file:9d2d55843e86c79ec0db8291fe7f516630a7f7fe2f0a90f2af552882cd94c7e9 in / "
        },
        {
            "created": "2025-08-01T17:33:19.437694874Z",
            "created_by": "/bin/sh -c #(nop) CMD [\"/bin/bash\"]",
            "comment": "FROM c50fc3f29ef2",
            "empty_layer": true
        }
    ]
}
```

Example 12-3 Inspect an image in an insecure private registry

```
skopeo inspect --tls-verify=false docker://localhost:5000/myimage:v1
```

Copying an Image

Use the `skopeo copy` command to copy container images between container storage types, such as local storage and container registries.

Use the `skopeo copy` command to copy an image between registries without needing to download it locally first. You can also copy an image to local Podman container storage.

For more information about the `skopeo copy` command, see the `skopeo-copy(1)` manual page.

Example 12-4 Copy an image from one registry to another

```
skopeo copy docker://container-registry.oracle.com/os/oraclelinux:9-slim
docker://myregistry.example.com:5000/oraclelinux:9-slim
```

Example 12-5 Copy an image from a registry to a private insecure registry

To copy an image to a private insecure registry, include the `--dest-tls-verify=false` option.

```
skopeo copy docker://container-registry.oracle.com/os/oraclelinux:9-slim --
dest-tls-verify=false docker://localhost:5000/ol9image:v1
```

Example 12-6 Copy an image from private secure registry to another registry

To copy an image from a private insecure registry, include the `--src-tls-verify=false` option.

```
skopeo copy --src-tls-verify=false docker://localhost:5000/ol9image:v2
docker://myregistry.example.com:5000/oraclelinux:v2
```

Example 12-7 Copy an image from a registry to the local storage

To copy an image to the local storage, use the `containers-storage:` prefix for the destination image.

```
skopeo copy docker://container-registry.oracle.com/os/oraclelinux:9-slim
containers-storage:oraclelinux:9-slim
```

Example 12-8 Copy an image from local storage to an insecure private registry

To copy an image from the local storage, use the `containers-storage:` prefix for the source image.

```
skopeo copy containers-storage:localhost/oraclelinux:9-slim --dest-tls-
verify=false docker://localhost:5000/ol9image:v2
```

Example 12-9 Copy an image to the local storage extract it

To download an image and review its internal content offline, specify a directory with the `dir:` prefix. For example, to extract the `oraclelinux:9-slim` image to the `oraclelinux` folder in a home directory:

```
skopeo copy docker://container-registry.oracle.com/os/oraclelinux:9-slim dir:/
home/$USER/oraclelinux
```

The `oraclelinux` folder contains a `manifest.json` file and several tarballs representing the copied image.

Synchronizing Images

Use the `skopeo sync` command to synchronize images between storage locations. This can be useful to create and keep container mirrors synchronized in an air gapped environment.

You can copy images from one image storage location to another to synchronize images using the `skopeo sync` command. The images are copied from the source to the destination using a specified transport. The transport must be specified in the command using one of the following transport types:

- `docker`: A container registry. This can be used for either the source or destination. If no image tag is specified, all the tags found for the image are copied.
- `dir`: A local directory path. This can be used for either the source or destination. When used with the destination, a directory is created for each image that's copied.
- `yaml`: A YAML file that contains the list of images to be copied from one container registry to another. This can't be used with the `dir` transport for local file systems.

The `--scoped` options adds the name of the source as a prefix to images so that many images with the same name can be stored at the destination image storage type (registry or directory).

Include the `--dry-run` option with the `skopeo sync` command to perform a dry run.

The format for a YAML file to use when synchronizing registry images lists the images to copy from one or more registries. For example:

```
myregistry.example.com:
  images:
    oraclelinux:
      - "9"
      - "10"
localhost:5000:
  images:
    myimage: []
  tls-verify: false
```

For more information on the `skopeo sync` command and the options that can be included in the YAML file, see the `skopeo-sync(1)` manual page.

Example 12-10 Synchronize an image from an insecure registry to local disk

Copy all the images named `oraclelinux` from a local insecure registry to a local directory.

```
skopeo sync --src docker --dest dir --scoped --src-tls-verify=false
localhost:5000/oraclelinux $HOME/images/
```

Example 12-11 Synchronize images from the local disk to a container registry

Copy all the images in a local directory to a registry:

```
skopeo sync --src dir --dest docker --scoped $HOME/images/localhost:5000
myregistry.example.com:5000/
```

Example 12-12 Synchronize images in a registry to an insecure private registry

Copy all the images named `oraclelinux` in a registry to a local private registry. The `--append-suffix` option adds a suffix to the destination image tag.

```
skopeo sync --src docker --dest docker --dest-tls-verify=false --append-
suffix '-mirror' myregistry.example.com:5000/oraclelinux localhost:5000/
mirror/
```

Example 12-13 Synchronize images using a YAML file

Copy the images listed in a YAML file named `mysync.yaml` to an insecure private registry.

```
skopeo sync --src yaml --dest docker --dest-tls-verify=false mysync.yaml
localhost:5000/mirror/
```

Deleting an Image

Use the `skopeo delete` command to delete an image from a container registry or from local container storage.

The `skopeo delete` command might not work with every container registry, and many registries don't allow this in the default configuration, or at all.

For more information about the `skopeo delete` command, see the `skopeo-delete(1)` manual page.

Example 12-14 Delete an image from local storage

```
skopeo delete containers-storage:oraclelinux:9-slim
```

Example 12-15 Delete an image from a private registry

```
skopeo delete docker://myregistry.example.com/oraclelinux:9
```

Example 12-16 Delete an image from a private insecure registry

```
skopeo delete --tls-verify=false docker://localhost:5000/myimage:v1
```

13

Container Registries

Use the container registries configured with Podman to pull container images. Create a self-hosted container registry, and create registry mirrors.

A container registry is a store of Open Container Initiative images. A container image is a read-only template which is used to create running containers. Container images in the registry can be deployed as required.

By default, Oracle Linux systems are configured with access to these commonly used registries:

- Oracle Container Registry (container-registry.oracle.com)
- Docker Hub (docker.io)

You can configure the container runtime to only trust images from container registries if they're signed to improve security and mitigate against inadvertently running a compromised image. For more information, see Configuring Podman for Signed Images.

Enterprise environments might also consider setting up a local container registry. The local container registry can store images converted from customized containers. You can then use these images for future container deployment. Storing images in a local container registry reduces the amount of customized configuration that you might need to perform for mass deployments. A local registry can also cache and host images that are pulled from an upstream registry, which further reduces network overhead and latency when you deploy matching containers across a spread of local systems.

Oracle Container Registry

Use the Oracle Container Registry to pull open source and licensed Oracle software container images.

The Oracle Container Registry is an open standards-based, Oracle-managed container registry service for securely storing and sharing container images. The Oracle Container Registry contains both licensed and open source Oracle software, and the images are built and signed by Oracle. Use of the container images is subject to the terms of their respective licenses.

The Oracle Container Registry is at https://container-registry.oracle.com. It provides a web interface to browse and select the images for the software that an organization can use. Oracle Container Registry hosts container images for both open source and licensed Oracle products. Licensed content requires you to authenticate and accept license agreement terms before downloading images. Before you can sign in, you must generate an authentication token.

> **Note**
>
> You don't need to create an authentication token and sign in to Oracle Container Registry to pull open source container images, as this is only required for licensed images.

You can use one of the Oracle Container Registry mirrors for faster download in different geographical regions.

Pulling Open Source Software From the Oracle Container Registry

The correct command to pull an image is provided on the repository information page in the Oracle Container Registry web interface. Other useful information about the image and how to run it might also be available on the same page.

If you're using an Oracle Container Registry mirror, change the URL in these steps to use the URL of the mirror. For more information on using mirrors, see Using Oracle Container Registry Mirrors.

To pull an image from the Oracle Container Registry, use the following `podman pull` command:

```
podman pull container-registry.oracle.com/area/image:tag
```

Replace *area* with the repository location in the Oracle Container Registry, and *image* with the name of the software image. For example:

```
podman pull container-registry.oracle.com/os/oraclelinux:9-slim
```

The *area* and *image* are specified in lowercase. When referencing images, we recommend that you always specify the appropriate *tag* to use.

For more information on the tagging conventions for Oracle Linux images, see Oracle Linux Container Image Tagging Conventions.

Pulling Licensed Software From the Oracle Container Registry

The Oracle Container Registry contains images for licensed commercial Oracle software products. To pull images for licensed software on the Oracle Container Registry, you must have an Oracle Account and have created an authentication token. You can create an Oracle Account at https://profile.oracle.com/myprofile/account/create-account.jspx. For information on creating an authentication token, see Generating an Oracle Container Registry Authentication Token.

> **Note**
>
> You don't need to sign in to the Oracle Container Registry or accept the Oracle Standard Terms and Restrictions to pull open source Oracle software images, only for licensed software images.

If you're using an Oracle Container Registry mirror, change the URL in these steps to use the URL of the mirror. For more information on using mirrors, see Using Oracle Container Registry Mirrors.

1. Sign in to the Oracle Container Registry.

   In a web browser, sign in to the Oracle Container Registry using an Oracle account at https://container-registry.oracle.com.

2. Accept the Oracle Standard Terms and Restrictions.

Accept the Oracle Standard Terms and Restrictions for the Oracle software images you want to pull. Acceptance of these terms are stored in a database that links the software images to an Oracle Account. Acceptance of the Oracle Standard Terms and Restrictions is valid only for the repositories for which you accept the terms. You might need to repeat this process if you try to pull software from other or newer repositories in the registry. Note that Oracle Standard Terms and Restrictions are subject to change without notice.

3. Find the image.

   Browse or search for Oracle software images in the Oracle Container Registry.

4. Sign in to the Oracle Container Registry in Podman.

   On the host system, use the `podman login` command to authenticate against the Oracle Container Registry.

   ```
   podman login container-registry.oracle.com
   ```

   Provide an Oracle account username, and its associated Oracle Container Registry secret key as the password value, when prompted.

5. Pull the image.

   Pull the images that you require by using the `podman pull` command, for example:

   ```
   podman pull container-registry.oracle.com/java/serverjre
   ```

   The image is pulled from the Oracle Container Registry and stored locally, ready to be used to deploy containers.

6. Sign out of the Oracle Container Registry in Podman.

   After you have pulled images from the Oracle Container Registry, log out of the registry to prevent unauthorized access and to remove any record of sign in credentials that Podman might store for future operations:

   ```
   podman logout container-registry.oracle.com
   ```

Generating an Oracle Container Registry Authentication Token

To pull licensed software from the Oracle Container Registry, generate an authentication token and use it as the `password` value when using the `podman login` command.

> ⚠ **Caution**
>
> From 2025-06-30 onward, authentication tokens are the only accepted credential type when authenticating Podman with the Oracle Container Registry.

1. Sign in to the Oracle Container Registry.

   In a web browser, sign in to the Oracle Container Registry using an Oracle account at https://container-registry.oracle.com.

2. Select the profile name.

   Select the profile name, and in the profile menu that appears select **Auth Token**.

3. Generate the Secret Key.

Select **Generate Secret Key** and note down the secret key. This is only displayed once, during the initial generation.

4. (Optional) Regenerate the Secret Key.

   If you lose or forget the secret key, generate a new one by selecting **Delete Secret Key**, then select **Generate Secret Key** again.

Using Oracle Container Registry Mirrors

The Oracle Container Registry has many mirror servers around the world. You can use a registry mirror in a specific global region to improve download performance of container images.

To list all the available mirrors and the command to use for pulling images from a specific mirror, see the information page for an image on the Oracle Container Registry web interface. The **Tags** table on the information page includes a **Download Mirror** drop down list to select a registry mirror. When you select a mirror, the **Pull Command** column changes to show the command to pull the image from the selected mirror.

To download licensed Oracle software images from a registry mirror, you must first accept the Oracle Standard Terms and Restrictions in the Oracle Container Registry web interface, and sign in to the Oracle Container Registry in Podman. For information on pulling licensed software images, see Pulling Licensed Software From the Oracle Container Registry.

If you use a mirror regularly, add it to the configuration so that the mirror is used by default for searches and pull requests. See Configuring Registries for more information.

Example 13-1 Pull an image from a mirror

Pull the Oracle Linux 9 slim image from the Sydney mirror:

```
podman pull container-registry-sydney.oracle.com/os/oraclelinux:9-slim
```

Example 13-2 Pull a licensed image from a mirror

To pull licensed Oracle software images, sign in to the Oracle Container Registry mirror with Oracle SSO username and Oracle Container Registry authentication token as the password before you pull the image. We recommend you sign out of the registry after the image is pulled.

For example, to pull the latest Java Platform, Standard Edition (Java SE) image:

```
podman login container-registry-sydney.oracle.com


podman pull container-registry-sydney.oracle.com/java/serverjre:latest


podman logout container-registry-sydney.oracle.com.oracle.com
```

Oracle Linux Container Image Tagging Conventions

Oracle follows several conventions when tagging container images for Oracle Linux. Users should be aware of these conventions to ensure that the best image is used for the purpose at hand to avoid unnecessary breakages in functionality and to help ensure that images continue to use the most recently patched software.

The slim Tag

Oracle releases minimal compressed versions of each Oracle Linux release. These images contain enough of the OS to run within a container and to perform installations of more packages. These images are the recommended images for general use within builds and where scripted installation is likely to be used. The images that use this tag are maintained at the most current update level.

For example, to use the most recent version of an Oracle Linux 9 slim image, use the `9-slim` tag. To use the most recent version of an Oracle Linux 10 slim image, use the `10-slim` tag.

FIPS compliant versions of images are tagged with the `slim-fips` tag. These images include compliant cryptographic package versions and most of the initial image setup required for container FIPS compliance. To use these images, you must enable FIPS mode on the host system.

Example 13-3 Pull an Oracle Linux 9 slim image

Pull the most recent version of an Oracle Linux 9 slim image:

```
podman pull oraclelinux:9-slim
```

General Oracle Linux Release Tags

Oracle Linux images are tagged at their release level and are maintained to always map to the latest corresponding update level. If you need a more complete OS than the version provided in a slim image, use a release tag to obtain the latest image for that Oracle Linux image.

Example 13-4 Pull the latest Oracle Linux 9 image

Pull the latest update release image for Oracle Linux 9:

```
podman pull oraclelinux:9
```

Oracle Linux Update Level Tags

Oracle Linux images are tagged at their update level. The other tags described map onto the latest or most current update level for an Oracle Linux image.

> **Warning**
>
> Don't directly use update level tags within a Containerfile or within any builds unless you have a specific use case that requires a particular update level. Typical use cases involve trying to resolve an issue or bug that's only present at a particular update level of Oracle Linux.
>
> Using an update level tag can result in containers running unpatched software that might expose them to security issues and software bugs.

Update level tags use dot notation to indicate the update level. For example, Oracle Linux 8.10 is indicated using the `8.10` tag:

```
podman pull oraclelinux:8.10
```

The latest Tag

> **Important**
>
> Oracle doesn't provide this tag for Oracle Linux images. Use a slim image or a release tag instead. Oracle also recommends that users avoid dependency on this tag when working with other distribution or software images.

The use of a default often results in significant confusion and can break builds and scripted functionality for end users. For this reason, and to help encourage best practice when working with image tags, Oracle doesn't provide a `latest` tag for Oracle Linux images.

The following reasons for Oracle's decision on this help explain why this tag isn't available:

- When the `latest` tag is used, it can result in significant jumps between distribution releases rather than update levels. This is often not what a user intends when selecting the `latest` tag, or depending on tools to fall back to this tag by not specifying a tag at all. Expected functionality can change dramatically between releases resulting in changes to commands, options, configurations, and available software.
- No easy way to identify which `latest` image was used for a particular build exists, making it difficult to see the differences between two final build images. This problem tracking changes also makes it difficult to roll back to a known functioning base image if a new build fails.
- Tagging an image with the `latest` tag isn't automatic and it's possible for a more recent image to be available while the image tagged as `latest` hasn't been updated. This can lead to unexpected consequences.
- Not all tools treat the `latest` tag the same. While some tools might default to always pulling an image tagged as `latest` from an upstream registry, other tools might default to a locally stored image also tagged as `latest`, even if it has fallen out of date.

This decision might result in errors in some tools that fall back to the `latest` tag when no tag is specified for an image. For example, the following command returns an error:

```
podman pull docker.io/library/oraclelinux
```

The error looks similar to:

```
Trying to pull docker.io/library/oraclelinux:latest...
Error: initializing source docker://oraclelinux:latest: reading manifest
latest in
docker.io/library/oraclelinux: manifest unknown
```

Always specify the appropriate tag for the image that you intend to use. For example:

```
podman pull container-registry.oracle.com/os/oraclelinux:9
```

Docker Hub

Use the Docker Hub Registry to pull software container images.

The Docker Hub registry provides many software images, primarily for use with Docker but which are compatible with Podman. The Docker Hub registry provides a web interface for browsing available images.

The Docker Hub contains Docker images for licensed commercial Oracle software products that you might use in an enterprise.

To access images hosted in the Docker Hub, you must sign in with a valid Docker account. To get more information about creating a Docker account and using the Docker Hub, see the Docker Hub documentation.

The Docker Hub provides a web interface where you can select the Docker Certified images that you want to install. For some images, you would need to select **Proceed to Checkout** to either agree to any terms and conditions that might apply or to make payment if required before you can access the image.

At the conclusion of the transaction, the image is stored in the **My Content** area, which you can revisit later. Each image provides a description and set up instructions.

Use the `podman login` command to sign in to Docker Hub and pull images. For example:

```
podman login docker.io


podman pull docker.io/library/alpine:latest


podman logout docker.io
```

Configuring Podman for Signed Images

Set up Podman to pull images that are signed by a validated public key for a container registry. Any unsigned images can't be pulled.

You can configure Podman to only trust images from a remote registry if they're signed and the provided signature can be validated against a locally stored public key. This configuration option can help improve security and can mitigate against inadvertently running a compromised image.

Images are signed in a similar way to packages that are made available on the Oracle Linux yum server. GPG keys are used to sign images provided at the registry. The digital signatures for each image are stored in a signature store that's accessible by using HTTPS. A public GPG key that's used to validate the signature against the image digest must be available on the system where Podman is installed.

The following steps describe how to configure a Podman host to require that images from a remote registry are signed and validated before they can be used locally.

1. Create a registry configuration file.

   For each registry where you require signature validation, create a YAML format configuration file in the `/etc/containers/registries.d/` directory (or

in `$HOME/.config/containers/registries.d/` directory for standard users) and provide the value for the `sigstore` for that registry.

For example, for the Oracle Container Registry, create the `/etc/containers/registries.d/oracle.yaml` file and populate it with the following content:

```
docker:
  container-registry.oracle.com:
    sigstore: https://container-trust.oci.oraclecloud.com/podman
```

See `/etc/containers/registries.d/default.yaml` for more information and to view a template configuration.

2. Download the public GPG key.

   Download and store the public GPG key that must be used to validate signatures for images from the registry. For the Oracle Container Registry, you can download the public GPG key from https://container-trust.oci.oraclecloud.com/podman/GPG-KEY-oracle. For example:

   ```
   sudo mkdir -p /etc/pki/containers
   ```

   ```
   sudo wget -O /etc/pki/containers/GPG-KEY-oracle https://container-trust.oci.oraclecloud.com/podman/GPG-KEY-oracle
   ```

3. Set the container policy.

   Edit the container policy configuration to add the location of the public GPG key that must be used to validate the signatures for images that are pulled from a particular registry.

   The policy configuration is in JSON format and is at `/etc/containers/policy.json`. Registry configuration appears under the `docker` key, which you might need to add under the `transports` key in the existing configuration. For example, a default policy configuration that has been edited to include an entry for the Oracle Container Registry appears as follows:

   ```
   {
     "default": [
       {
         "type": "insecureAcceptAnything"
       }
     ],
     "transports":
       {
         "docker-daemon":
           {
             "": [{"type":"insecureAcceptAnything"}]
           },
         "docker":
           {
             "container-registry.oracle.com": [
               {
                 "type": "signedBy",
                 "keyType": "GPGKeys",
                 "keyPath": "/etc/pki/containers/GPG-KEY-oracle"
               }
   ```

```
            ]
        }
    }
}
```

See the `containers-policy.json(5)` manual page for more information about the format of this configuration file.

4. Validate the configuration.

   Validate that the configuration is correct by pulling an image from the remote registry. For example:

   ```
   podman pull container-registry.oracle.com/os/oraclelinux:9-slim
   ```

   If the signature requirement is configured correctly, the output isn't different from an output when you pull an image without signature validation configured.

5. (Optional) Confirm GPG key is used.

   You can test that validation is taking place by setting the GPG `keyPath` in the policy configuration to use another key. For example, you can configure the path to use the GPG key used to validate RPM packages at `/etc/pki/rpm-gpg/RPM-GPG-KEY-oracle`. When pulling the same image, validation of the signature fails with an error similar to:

   ```
   Trying to pull container-registry.oracle.com/os/oraclelinux:9-slim...
   Error: copying system image from manifest list: Source image rejected:
   Invalid GPG signature:
   gpgme.Signature{Summary:128,
   Fingerprint:"4451CA2B13A2D6522D9E12DF357217938FC350A2",
   Status:gpgme.Error{err:0x9}, Timestamp:time.Date(2025, time.July, 16, 19,
   18, 22, 0, time.Local),
   ExpTimestamp:time.Date(1970, time.January, 1, 0, 0, 0, 0, time.Local),
   WrongKeyUsage:false,
   PKATrust:0x0, ChainModel:false, Validity:0, ValidityReason:error(nil),
   PubkeyAlgo:1, HashAlgo:8}
   ```

Configuring Registries

Edit the container registry configuration file to set registry options for the environment. This can be set system wide, and for users.

To configure default registry settings, edit the `/etc/containers/registries.conf` file, which is in TOML format. The configuration file is commented to explain the options that are available. You can create a user specific registry configuration file by copying the system wide configuration file (`/etc/containers/registries.conf`) to a user's directory at `$HOME/.config/containers/registries.conf`. For example:

```
cp /etc/containers/registries.conf $HOME/.config/containers/registries.conf
```

For detailed information on all options in the container registries configuration file, see the upstream documentation, or the `containers-registries.conf(5)` manual page.

Listing Registries

To see the registries configured to use when searching for container images, use:

```
podman info --format='{{.Registries}}'
```

The output looks similar to:

```
map[search:[container-registry.oracle.com docker.io]]
```

Setting Registry Order

The registries that are searched when you try to pull or use an image that isn't available locally are defined in the following configuration block:

```
unqualified-search-registries = ["container-registry.oracle.com", "docker.io"]
```

Registries are searched sequentially in the order that they're defined in this list. If a local registry exists, add it at the beginning of the list to make it the first searched registry.

Adding Registries

To add a registry to Podman, edit the Podman system wide or user configuration file to include the registry in the `unqualified-search-registries` entry. Add it in the order you want container searches and pulls to be performed. For example:

```
unqualified-search-registries = ["localhost:5000",
"myregistry.example.com:5000", container-registry.oracle.com", "docker.io"]
```

If the registry is an insecure registry, also edit the `[[registry]]` section and include the `insecure = true` option with the registry location.

```
[[registry]]
location = "localhost:5000"
insecure = true
```

Adding Insecure Registries

To use an insecure registry without a valid SSL certificate or that doesn't use SSL, edit the `[[registry]]` section and include the `insecure = true` option with the registry location. For example:

```
[[registry]]
location = myregistry.example.com:5000
insecure = true

[[registry]]
location = "localhost:5000"
insecure = true
```

14

Private Container Registries

Set up a local, private container registry server to host container images within an organization. A private registry can also be used to mirror the Oracle Container Registry.

The registry server is a container application. The host must have an Internet connection to download the registry image from Docker Hub or, if support is required, from the Oracle Container Registry.

You can create a secure private registry, which requires you to set up TLS. When you set up TLS, you can use a self signed Certificate Authority (CA) certificate, or a certificate signed by a CA. Otherwise, the registry can be created as an insecure registry for testing and development purposes.

> **Note**
>
> The registry image from Docker Hub can be used to set up both an insecure registry and a secure registry. The registry image from the Oracle Container Registry can only be used to set up a secure registry.

Creating an Insecure Registry

Create a local container registry without TLS. This can be used for testing or development. A secure registry that uses TLS is recommended for a production system.

If you create an insecure registry, you can add it to the Podman registry configuration file to avoid using the `--tls-verify=false` option when using the registry. For more information, see Adding Insecure Registries.

1. Create the registry.

   Use the `podman create` command to create the registry. For example:

   ```
   sudo podman run -d -p 5000:5000 --name registry --restart always registry:2
   ```

   Or for a standard user:

   ```
   podman run -d -p 5000:5000 --name registry --restart always registry:2
   ```

   The `-d` option runs the container in the background.

   The `-p` option publishes the port mapping. If you're using the default ports, set this to `5000:5000`. If port 5000 is already in use, assign another port, for example use `5001:5000`.

   The `--name` option sets the name of the registry container. This is commonly set to `registry`.

The container registry image is pulled from Docker Hub using the short name of `registry:2`, though you could also use the full image location of `docker.io/library/registry:2`.
For more information on the options, use the `podman run --help` command.

2. Verify the registry container is running:

   Use the `podman ps` command to verify the registry container is running.

   ```
   sudo podman ps
   ```

   Or for a standard user:

   ```
   podman ps
   ```

Creating a Secure Registry

Create a local container registry with TLS. A secure registry that uses TLS is recommended for a production system.

Before you begin, ensure you have considered the prerequisites:

- The registry server requires at least 15 GB of available disk space to store registry data in `/opt/registry/`.
- As a good practice, create a separate file system for the registry, preferably a Btrfs formatted file system. By using the Btrfs file system, you can easily scale the registry file system and leverage Btrfs features such as file system snapshots. For information on setting up Btrfs, see the following documents:
  - Oracle Linux 8: Managing the Btrfs File System
  - Oracle Linux 9: Managing the Btrfs File System
  - Oracle Linux 10: Managing the Btrfs File System
- The registry host requires a valid X.509 certificate and private key to enable Transport Layer Security (TLS) with the registry, similar to using TLS for a web server. You can use either a certificate signed by a trusted Certificate Authority (CA), or a self signed certificate. A self signed certificate can be used for testing purposes. For information about creating a self signed certificate and private key, see Oracle Linux: Managing Certificates and Public Key Infrastructure.

  If the host already has an X.509 certificate, you can use it with Podman to create the private registry. If you're using the host's X.509 certificate and it was issued by an intermediate CA, combine the host's certificate with the intermediate CA's certificate to create a chained certificate. This creates a certificate that includes the both the host and intermediate CA certificate so it can be validated by Podman. Use the `cat` command to do this, using the format:

  ```
  sudo cat host_certificate.crt intermediate_certificate.pem >
  chained_certificate.crt
  ```

  For example:

  ```
  sudo cat myregistry.example.com.crt intermediate-ca.pem > domain.crt
  ```

1. Create the required directories.

   The `/opt/registry/` directory is used to store the registry data. The `certs` directory stores the certificate and private key, and the `data` directory is used to store the images pushed to the registry.

   ```
   sudo mkdir -p /opt/registry/{certs,data}
   ```

2. Copy the certificate and key to the `/opt/registry/certs/` directory.

   Use the format:

   ```
   sudo cp certfile /opt/registry/certs/domain.crt
   ```

   ```
   sudo cp keyfile /opt/registry/certs/domain.key
   ```

   Replace *certfile* with the full path to the host's X.509 certificate, or to the chained certificate. Replace *keyfile* with the full path to the host's private key. For example:

   ```
   sudo cp /etc/pki/tls/certs/registry.example.com.crt /opt/registry/certs/
   domain.crt
   ```

   ```
   sudo cp /etc/pki/tls/private/registry.example.com.key /opt/registry/certs/
   domain.key
   ```

3. Set private key permissions.

   Set the correct file permissions for the private key.

   ```
   sudo chmod 600 /opt/registry/certs/domain.key
   ```

4. Create the registry.

   Use the `podman create` command to create the registry. For example:

   ```
   sudo podman run --name registry \
   -d \
   -p 5000:5000 \
   -v /opt/registry/data:/var/lib/registry:z \
   -v /opt/registry/certs:/certs:z \
   -e "REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt" \
   -e "REGISTRY_HTTP_TLS_KEY=/certs/domain.key" \
   -e REGISTRY_AUTH="" \
   registry:2
   ```

   The `--name` option sets the name of the registry container. This is commonly set to `registry`.

   The `-d` option runs the container in the background.

   The `-p` option publishes the port mapping. If you're using the default ports, set this to `5000:5000`. If port 5000 is already in use, assign another port, for example use `5001:5000`.

   The `-v` options mount local directories in the container.

Set environment variables to use with the registry container using the `-e` option. Use this option to set the location of the X.509 certificate and key information.

The container registry image is pulled from Docker Hub using the short name of `registry:2`, though you could also use the full image location of `docker.io/library/registry:2`. You can also set this to pull from the Oracle Container Registry by setting the container to `container-registry.oracle.com/os/registry:latest`.

For more information on the options, use the `podman run --help` command.

5. Verify the registry container is running:

   Use the `podman ps` command to verify the registry container is running.

   ```
   sudo podman ps
   ```

6. Open the registry port in the firewall.

   If you're running a firewall, ensure the TCP port you want the container registry to listen on is accessible. For example:

   ```
   sudo firewall-cmd --zone=public --permanent --add-port=5000/tcp
   ```

   ```
   sudo firewall-cmd --zone=internal --permanent --add-port=5000/tcp
   ```

   Reload the `firewalld` configuration:

   ```
   sudo firewall-cmd --reload
   ```

7. Configure the registry with Podman.

   Add the container registry to the Podman configuration file on each host that needs to access the registry. For information on configuring registries, see Configuring Registries.

Distributing X.509 Certificates

If the registry host uses a self-signed X.509 certificate, you must distribute the certificate to **all** the hosts in the deployment for which you intend to use the local container registry.

For the root user, certificates for each registry are stored in `/etc/containers/certs.d/`*`registry_hostname`*`:`*`port`*`/`. For standard users, certificates can be stored in `$HOME/.config/containers/certs.d/`*`registry_hostname`*`:`*`port`*`/`. Change *registry_hostname* to the name of the registry host, and *port* to the port number for the container registry server (5000 by default).

Podman, Buildah, and Skopeo commands that interact with registries also often provide a `--cert-dir` option to specify an alternate location for these certificates.

1. Create a `certs.d` directory.

   Create the appropriate `certs.d` location for the registry host and the user. For the root user, use the format:

   ```
   sudo mkdir -p /etc/containers/certs.d/registry_hostname:port
   ```

For a standard user, use the format:

```
mkdir -p $HOME/.config/containers/certs.d/registry_hostname:port
```

Replace *registry_hostname* with the name of the server running the container registry. Replace *port* with the port number to access the registry. For example:

```
sudo mkdir -p /etc/containers/certs.d/myregistry.example.com:5000
```

Or for the standard user:

```
mkdir -p $HOME/.config/containers/certs.d/myregistry.example.com:5000
```

2. Copy the X.509 certificate from the registry host.

```
sudo scp user@registry_hostname:/opt/registry/certs/domain.crt \
/etc/containers/certs.d/registry_hostname:port/ca.crt
```

Or for the standard user:

```
scp user@registry_hostname:/opt/registry/certs/domain.crt \
$HOME/.config/containers/certs.d/registry_hostname:port/ca.crt
```

Importing Images Into a Registry

When you have set up a private container registry, you can import images into the registry so that they can be used to deploy containers. You can pull images from a registry, such as the Oracle Container Registry, and then commit them to a local registry. You can also create custom images based on upstream images.

1. Pull an image from a registry.

   For example, pull an image from the Oracle Container Registry:

```
podman pull container-registry.oracle.com/os/oraclelinux:9-slim
```

2. Tag the image.

   Tag the image so that it points to the local registry. For example:

```
podman tag container-registry.oracle.com/os/oraclelinux:9-slim
myregistry.example.com:5000/ol9image:v1
```

   In this example, `myregistry.example.com:5000` is the location of the private registry. Change this to the location of the private registry. If you're using an insecure registry on the same host, you can use `localhost:5000` for the registry location. The repository and tag name, `ol9image:v1` in this example, must all be in lowercase to be a valid tag.

3. Push the image to the registry.

   Push the image to the registry, for example:

```
podman push myregistry.example.com:5000/ol9image:v1
```

If you're pushing to an insecure registry, include the `--tls-verify=false` option.

4. Verify the image is loaded.

   Check the image is loaded to the container registry by pulling the image.

   First, remove the existing local image pushed in the previous step.

   ```
   podman image rm ol9image:v1
   ```

   Pull the image from the registry:

   ```
   podman pull myregistry.example.com:5000/ol9image:v1
   ```

   Verify the local image pulled from the registry:

   ```
   podman images
   ```

See Buildah for more information about how you can create images. When you have committed a customized image, you can tag it and push it to the local registry.

15

Podman Command Reference

The following list includes the `podman` command subcommands.

Each of the listed commands is linked to a manual page that follows the `podman-`*`command`*`(1)` pattern. For example, to retrieve information about the `attach` command, see the `podman-attach(1)` manual page. For a full list of all the documentation available for Podman, see the `podman(1)` manual page. You can also use the `podman help` command to get more information on the command syntax.

artifact
Manage Open Container Initiative artifacts. Not available on Oracle Linux 8 hosts.

attach
Attach to the shell of a running container.

auto-update
Automatically update containers with automatic updating enabled.

build
Build an image from a Containerfile.

commit
Create an image based on an edited container.

compose
Run compose workloads using an external provider such as `docker-compose` or `podman-compose`.

container
Manage existing containers.

cp
Copy files and folders between the container and host file system.

create
Create a container without starting it.

diff
View changes on the container's file system.

events
Show Podman event logs for the running container.

exec
Run a process in a specified container that's already running.

export
Export a container's file system and contents to a compressed archive.

farm
Farm out builds to remove machines that Podman can connect to using a system connection.

generate
Generate `systemd` unit files and pod YAML files.

healthcheck
Run a health check on an existing container.

history
Review the history of a specified image.

image
Manages existing images.

images
Lists images present on the host system.

import
Import a compressed archive to create a container file system.

info
Show system information for Podman.

init
Initialize one or more containers.

inspect
Display the existing configuration values for a container or image.

kill
Stop running containers with a predefined signal.

kube
Generate, and play containers, pods, or volumes using a structured Kubernetes compliant YAML file.

load
Load an image from a container archive file.

login
Sign in to a container registry.

logout
Sign out of a container registry.

logs
Review logs for a container.

machine
Manage a virtual machine.

manifest
Create and edit manifest lists and image indexes.

mount
Mount a running container's root file system.

network
Manage networks that are accessible to containers.

pause
Suspend all the processes in one or more containers.

play
Start containers, pods, or volumes from a structured file.

pod
Create and manage pods.

port
List network port mappings for a container.

ps
List containers.

pull
Pull an image from a container registry.

push
Push an image to a container registry.

rename
Rename an existing container.

restart
Restart one or more containers.

rm
Remove one or more containers.

rmi
Remove one or more images from local storage on the host system.

run
Run a single command in a new container.

save
Save an image to a compressed archive.

search
Search a container registry for an image.

secret
Manage secrets.

start
Start one or more containers.

stats
Display a real time stream of container resource usage statistics.

stop
Stop one or more containers.

system
Manage Podman configuration settings.

tag
Add another name to an image in local storage on the host system.

top
Display the running processes for a container.

unmount
Unmount a running container's root file system.

unpause
Resume all processes for one or more containers.

unshare
Run a command as a specified user.

untag
Remove a secondary name from an image in local storage on the host system.

update
Update an existing container.

version
Show version information for Podman.

volume
Manage container storage volumes.

wait
Block processes on one or more containers until a specified condition is fulfilled.

16

Security Recommendations

Ensure that infrastructure and containerized applications remain secure by following security recommendations and guidelines.

Follow security guidelines at all levels within the infrastructure for an environment to best mitigate against exploitation. Follow the best practice guidelines for each component at play within the environment.

While containerization provides resource separation between applications running on the same host, the separation isn't complete and it's possible to break out of a container or to exploit a container in such a way that it could affect other containers running on the same host. If you have different tenancies within the organization, or if you have different customers that are using the same infrastructure, it's imperative that their containers run on different hosts, or on different virtual machines, to achieve more complete separation and to prevent the likelihood of a serious data breach.

Host

Regularly update the host kernel and OS software
Security patches and bug fixes for the kernel and OS software are released as issues are resolved. Keep the OS current with the most recent software updates. Subscribe the system to the latest software channels or repositories, and run regular DNF update operations. Consider using Ksplice to keep the system software up-to-date.

Use a minimal OS and ensure that it follows security best practices
Where possible, use a minimal OS installation and ensure that it follows the security best practices described in the following documents:

- Oracle Linux 8: Enhancing System Security
- Oracle Linux 9: Enhancing System Security
- Oracle Linux 10: Enhancing System Security

Most importantly, reduce the number of services running on the same system. Ideally, move all other services to reside within containers controlled by Podman or move them to other systems entirely. This helps to contain damage in the event of container break out.

Regularly scrutinize the OS and kernel for safety
Be vigilant that the OS is regularly scrutinized for safety and potential vulnerabilities.

Use mature system components that provides the best possible security feature set
We recommend keeping all system components, including the kernel, up-to-date so that all known security and functionality issues have been resolved. If you're using the Unbreakable Enterprise Kernel (UEK), consider using the latest available UEK generation for the Oracle Linux release, so that kernel-based features such as kernel namespaces, private networking, and control groups are mature, reliable, and heavily tested.

Use Linux security modules
Use the appropriate security modules on a host where possible. For example, run SELinux in enforcing mode to protect the host from the container images you're running. In addition, when

using a volume mount, consider using the `-z` option if you need to share an SELinux security context between containers or between the container and the host system. This is useful when running a container as a non root user because the container is evaluated with the same SELinux context as the host. To restrict the volume to only the running container, use the `-Z` option. See Setting Up Container Mounts for more information.

Podman Images

Ensure that images come from verified and trusted sources
Verify that Podman images are received and deployed unchanged from a source with a trusted reputation and which has been authenticated. For example, see Configuring Podman for Signed Images.
When pulling images from remote sources, ensure that the connection is protected and that you're using HTTPS for the pull request. Don't use insecure image registries that aren't protected by TLS.
Where possible, use Podman images based on a curated, trusted collection of image suppliers.

Create reliably reproducible images
When using Containerfiles to build new images, review base images, and installed software for security. To help ensure that new images use base images and software that you have reviewed for security vulnerabilities:

- Specify a fixed version in the base image in an image Containerfiles.
- Specify fixed versions in package pulls in the build steps of an image Containerfile (note that dependencies of dependencies can still be a reliability problem).
- Ensure the that package pulls in the build steps are using trusted and verified sources.

Minimize packages installed on images
Don't install unnecessary packages into new image builds. Review Containerfiles to remove unnecessary installation steps so that images remain limited to their function.

Podman Containers

Run containers as a non root user
Podman runs each container as the host user running the Podman container. The host user can be the root user or a non root user. For most security, run containers with a non root host user.

Consider running containers with limited memory and CPU usage
Consider limiting container memory and CPU usage using the `-m` and `--memory-swap` options for memory and swap memory, and the `-c` option for CPU.

Limit container restarts
To prevent potential denial-of-service resulting from a container that spins out of control, limit container restarts using the `--restart=on-failure:N` option when creating or running a container.

Monitor container resource usage
Podman provides facilities to monitor container resource usage, such as memory consumption, CPU time, I/O and network usage. Review container resource usage for performance, error detection, and anomalous behavior. Consider using tools to monitor real-

time resource usage for anomalous activity such as use of resources, suspicious traffic, and unexpected user activity.

Limit container file access
When creating and running containers, limit container file access using the `--read-only` flag or the `-v host dir:container dir:ro` option. Explicitly create volumes for container applications to write in and monitor changes to files in these volumes. Ensure that volumes that are dedicated for container write access are reviewed for sprawl and are cleaned up regularly. The following example shows how to use the :ro option to mount a host directory such that the host folder or file is read-only for the container:

```
podman run -v /host_directory_to_be_mounted:/target_container_directory:ro
oraclelinux:9
```

In the previous example, *host_directory_to_be_mounted* is the host directory and files to be mounted as a volume and *target_container_directory* is the directory on the container where the host directory is to be mounted.
Don't mount sensitive host system directories at container runtime:

- `/`
- `/boot`
- `/dev`
- `/etc`
- `/lib`
- `/proc`
- `/sys`
- `/usr`

Regularly review containers for safety
Consider using tools that help automate container safety checks and to monitor for changes within containers.
Systematically remove images and containers that aren't needed from the host system to avoid image and container sprawl and to help prevent the accidental usage of an old, unused image or container that has might have avoided security scrutiny. Consider using the Podman auto-update feature to automate the process of checking for new image versions and redeploying impacted containers automatically.

Understand kernel capabilities in containers
Oracle Linux divides the root user into distinct units, called capabilities. By default, the following kernel capabilities are *granted* to a container:

- `CHOWN`
- `DAC_OVERRIDE`
- `FSETID`
- `FOWNER`
- `NET_RAW`
- `SETGID`
- `SETUID`

- SETFCAP
- SETPCAP
- NET_BIND_SERVICE
- SYS_CHROOT
- KILL

By default, the following notable kernel capabilities are *removed* from a container:

- SYS_TIME
- MKNOD
- NET_ADMIN
- SYS_MODULE
- SYS_NICE
- SYS_ADMIN
- AUDIT_WRITE

Don't use the `--privileged` option when starting containers because it disables the security features that isolate the container from the host such as dropped capabilities, limited devices, read-only mount points, and volumes and so on.

Understand options to limit kernel file handle and process resources in containers
When creating and running containers, limit kernel resources by using the `--ulimit` option or set container defaults using the `--default-ulimit` when starting the Podman service.

Understand options to limit networking access from containers
If you do run a network for a container, when publishing ports to the host, specify the IP address of the interface that you want the port to bind to so that the attack surface is reduced to the network interface where the container is listening. Podman publishes to all interfaces (0.0.0.0) by default if an IP address isn't specified when using the `-p` or `--publish` option.
Don't run SSH inside containers.
Don't map privileged ports (< 1024) inside containers.
Don't use the `--net=host` mode option for containers when they're started or run. This option gives the container full access to local system services and is insecure.

Understand options to limit system calls in containers
By default, Podman containers limit the system calls available to containers using the system calls specified in the `/usr/share/containers/seccomp.json` file. This list is compatible with most containers and you don't need to add more system calls.

Don't share host namespaces with containers
Don't share host namespaces such as the PID or IPC namespaces when starting or running containers.

Don't expose host devices into containers
Don't expose host devices into containers when you start or run them.

Containerized Applications

Minimize kernel calls in containerized applications
Because the kernel is shared between containers, kernel calls increase risk to other containers running on the host system. Avoid kernel calls within containerized applications wherever possible.

Run Container applications as a non root user
Ensure that containerized applications run as a non root user. Because the UIDs are shared across the host, the root user in a container is the root user on the host.

Remove or minimize the use of `setuid` and `setgid` in containerized applications
Most applications don't need any `setuid` or `setgid` binaries. If you can, disable, or remove such binaries. By doing so, you remove the chance of them being used for privilege escalation attacks. If you discover binaries that have `setuid` or `setgid` permission flags, remove them altogether, or try to remove the permission flags to remove the risks that are associated with these permissions on a binary.

Design containerized applications to be impermanent
As much as is possible, design applications to be stateless, rollable, instantly migrateable microservices container apps if possible. If using applications outside of your own design, take this approach into consideration when selecting software that you intend to run within containers. This quality can be helpful in maintaining service during and in the time following a breach or accident in the system.

17

Known Issues

Ensure you read the known issues using Podman and related utilities.

Quadlets Fail For An Unprivileged User

Quadlets fail to run on Oracle Linux 8 without root permissions. While trying to start a Quadlet service, you might see the following error message:

```
Error: mkdir /sys/fs/cgroup/blkio/user.slice/runtime: permission denied
```

Or, if you're using `crun` as the runtime, you might see the following error message:

```
Error: OCI runtime error: crun: the requested cgroup controller `pids` is not
available"
```

Because of this problem, Podman Shell isn't available for Oracle Linux 8.

(Bug 36076771)

Podman Build Command Fails With "Operation Not Permitted" When Unprivileged Users Try to Establish Volumes

Oracle Linux 8 hosts running UEK 6 Update 3 or the Red Hat Compatible Kernel (RHCK) can't mount or umount volumes on rootless Podman containers. Typically, trying to do so causes this error message:

```
...
error running container: from /usr/bin/runc creating container for [/bin/sh -
c touch /tmp/myfile1]:
time="2024-02-06T00:22:07Z" level=warning msg="unable to get oom kill count"
error="no directory
specified for memory.oom_control" time="2024-02-06T00:22:07Z" level=error
msg="runc create failed:
unable to start container process: error during container init: error
mounting
\"/home/podman_user/.local/share/containers/storage/overlay-containers/
2f80607e49897c5a2a8020bfbc520ddf2ee48157722fe2e207b3491ada5df0c9/userdata/
overlay/1738053957/merge\"
to rootfs at \"/data\":
mount /home/podman_user/.local/share/containers/storage/overlay-containers/
2f80607e49897c5a2a8020bfbc520ddf2ee48157722fe2e207b3491ada5df0c9/userdata/
overlay/1738053957/merge:
/data (via /proc/self/fd/7),
data: lowerdir=/home/podman_user/.local/share/containers/storage/overlay/
fe866d78514c04dd5df86d3ff2fff3288c675a52874f114c36f7d94aa1666bd6/merged/data,
upperdir=/home/podman_user/.local/share/containers/storage/overlay-containers/
```

```
2f80607e49897c5a2a8020bfbc520ddf2ee48157722fe2e207b3491ada5df0c9/userdata/
overlay/1738053957/upper,
workdir=/home/podman_user/.local/share/containers/storage/overlay-containers/
2f80607e49897c5a2a8020bfbc520ddf2ee48157722fe2e207b3491ada5df0c9/userdata/
overlay/1738053957/work,
userxattr,context=\"system_u:object_r:container_file_t:s0:c10,c760\":
operation not permitted"
...
```

If that functionality is required, upgrade the Oracle Linux 8 host to boot from the UEK7. For more information, see the Unbreakable Enterprise Kernel documentation.

(Bug 36250501)

Container Storage is Not Accessible to an Unprivileged User

Container storage might fail to mount for an unprivileged user in some Oracle Linux 8 environments. Typically this displays in the following way:

```
$ podman run --name c_uidmap --uidmap 0:10000:10000
localhost:5555/os/oraclelinux:7 true
Error: error creating container storage: error creating an ID-mapped copy of
layer "a8c980a5275b9ef8dc35f68daacc8fc82e463cd25adeb84ccda98b58ce84f122":
exit status 1: error during chown: error mapping container ID pair
idtools.IDPair{UID:10000, GID:10000} for
"usr/lib64/python2.7/SimpleXMLRPCServer.pyo" to host: Container ID 10000
cannot be mapped to a host ID
```

The issue can be resolved by updating the container mount options to include the `index=off` parameter. Edit `/etc/containers/storage.conf` to make this change, for example:

```
mountopt = "nodev,metacopy=on,index=off"
```

(Bug 34161379)

X509 Certificate Relies on Legacy Common Name Field

With the release of Podman version 3.0, included with Oracle Linux 8.4, Podman commands that require TLS verification for certificates that don't include a proper Subject Alternative Name (SAN) return the following error:

```
x509: certificate relies on legacy Common Name field, use SANs or
temporarily enable Common Name matching with GODEBUG=x509ignoreCN=0
```

This issue is the result of a newer version of the Go compiler used to build Podman. The issue occurs when working with local or private image registries that use self-signed certificates.

You can either update certificates to use a proper SAN or set the GODEBUG environment variable `x509ignoreCN=0` in the environment where you intend to run Podman. For example

add the following line to `$HOME/.bashrc` file to continue using self-signed certificates where a SAN isn't set:

```
export GODEBUG=x509ignoreCN=0
```

(Bug 32821677)

Executing Podman Attach --latest Causes Panic if No Containers Are Available

If you run the `podman attach --latest` command and no containers exist in an environment, a runtime error similar to the following occurs:

```
sudo podman  attach --latest


panic: runtime error: index out of range
...
```

Note that this error no longer occurs as soon as containers exist in the environment. Running the command when no containers exist is meaningless.

(Bug 29882537)

Requirements for Using the Default Podman Detach Key Sequence

On Oracle Linux 8 hosts, the default key sequence that you use to detach a container (`CTRL+P`, `CTRL+Q`) requires a console that can handle detachment (`pseudo-tty`), and an input channel for passing control signals (`stdin`). Otherwise, you can't create a container, attach it with the `podman attach -l` command, and then quit, or detach the container by using the default key sequence, as documented in the `podman-attach(1)` manual page.

To ensure that you can use the default `CTRL+P`, `CTRL+Q` key sequence to detach a container, use either of the following methods to create a container. In both cases ensure that you use the `-t` option so that a `pseudo-tty` is created:

- Create a container in the background:

  ```
  sudo podman run --rm -dt container-registry.oracle.com/os/oraclelinux:8-
  slim top -b
  ```

  You can then use the `podman attach -l` command to attach the container and the `CTRL+P`, `CTRL+Q` key sequence to detach the container.

- Create a container interactively:

  ```
  sudo podman run --rm -it container-registry.oracle.com/os/oraclelinux:8-
  slim top -b
  ```

The interactive method creates the container and automatically attaches it. You can then use the `CTRL+P`, `CTRL+Q` key sequence to detach the container.

For more information, see the `podman(1)` and `podman-attach(1)` manual pages.

(Bug 29882852)

Authentication Error Occurs Pulling an Image Using an Incorrect Name

If you try to pull an image by running the `podman pull` *`image-name`* command, but you don't specify the correct or full name of the image, an authentication error occurs.

For example, the following error is displayed because `oracle:ol8-slim` was specified as the name of the image instead of `oraclelinux:ol8-slim`, which is the correct name for the image:

```
podman pull oracle:ol8-slim
```

Displays output similar to:

```
Trying to pull registry.redhat.io/oracle:latest...Failed
Trying to pull quay.io/oracle:latest...Failed
Trying to pull docker.io/oracle:latest...Failed
error pulling image "oracle:ol8-slim": unable to pull oracle:ol8-slim: 3
errors
occurred:

* Error determining manifest MIME type for
docker://registry.redhat.io/oracle:ol8-slim: unable to retrieve auth token:
invalid username/password
* Error determining manifest MIME type for docker://quay.io/oracle:ol8-slim:
Error reading manifest latest in quay.io/oracle: error parsing HTTP 404
response body: invalid character '<' looking for beginning of value:
"<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 3.2 Final//EN\">\n<title>404 Not
Found</title>\n<h1>Not Found</h1>\n<p>The requested URL was not found on the
server.  If you entered the URL manually please check your spelling and try
again.</p>\n"
* Error determining manifest MIME type for docker://oracle:ol8-slim: Error
reading manifest latest in docker.io/library/oracle: errors:
denied: requested access to the resource is denied
unauthorized: authentication required
```

To prevent this error from occurring, always specify the correct image name with the `podman pull` command.

(Bug 29894231)

The Latest Tag Is Missing From the oraclelinux Image on Docker Hub

Trying to search for or pull the `oraclelinux` image from Docker Hub fail with a *manifest unknown* error because the `latest` tag doesn't exist:

```
podman pull docker://docker.io/library/oraclelinux:latest
```

```
Trying to pull repository docker.io/library/oraclelinux ...
manifest for oraclelinux:latest not found: manifest unknown: manifest unknown
manifest for oraclelinux:latest not found: manifest unknown: manifest unknown
```

This issue also affects the Skopeo utility, which expects the `latest` tag to be present by default:

```
skopeo inspect docker://docker.io/library/oraclelinux
```

```
FATA[0001] Error parsing image name "docker://docker.io/library/oraclelinux":
Error reading manifest latest in docker.io/library/oraclelinux:
manifest unknown: manifest unknown
```

The `latest` tag was removed from the Oracle Linux official images in June 2020 to reduce customer confusion. Downstream images using `oraclelinux:latest` or no tag should be updated to `oraclelinux:8` for future builds. Note that we recommend using the `-slim` variants for the smallest possible image size.

For more information, see Oracle Linux Container Image Tagging Conventions.

(Bug 31524440)

Podman Pod Create Fails on Oracle Linux 9 For An Unprivileged User With IMA Enabled

On systems, such as Oracle Linux 9, where Integrity Measurement Architecture (IMA) is enabled and enforcing, running `podman pod create` as an unprivileged user can fail with an error similar to:

```
...
RemoveOptions:copier.RemoveOptions{All:false}}: copier: put: error setting
extended attributes on "/catatonit": error setting value of extended
attribute "security.ima" on "/catatonit": operation not permitted
```

This issue occurs when the `rpm-plugin-ima` package is installed before the `podman` or `podman-catatonit` packages. In the case where the `rpm-plugin-ima` package is installed the `catatonit` binary, used by Podman to provide init services to containers, is an IMA signed file and unprivileged users don't have permissions to set security extended attributes (xattrs) on the file system.

To work around the issue, uninstall `rpm-plugin-ima` and reinstall the `podman` package. Note that on fresh installations of Oracle Linux 9.2 and later, the `rpm-plugin-ima` package isn't installed by default and the issue is unlikely to appear. Also, from Podman version 4.4 onwards, the `catatonit` binary is included in the `podman` package and you don't need to install the `podman-catatonit` package.

(Bug 34578553)

Executing Podman Attach --latest Causes Panic if No Containers Are Available

If you run the `podman attach --latest` command and no containers exist in an environment, a runtime error similar to the following occurs:

```
sudo podman  attach --latest


panic: runtime error: index out of range
...
```

Note that this error no longer occurs as soon as containers exist in the environment. Running the command when no containers exist is meaningless.

(Bug 29882537)