Oracle Linux

Podman User's Guide

F30921-24
May 2024

Oracle Linux Podman User's Guide,

F30921-24

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.

Documentation License

Conventions

The following text conventions are used in this document:

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

Documentation Accessibility

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

Access to Oracle Support for Accessibility

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

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners,

we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

About Podman, Buildah, and Skopeo

Podman, Buildah, and Skopeo are a set of tools that you can use to create, run, and manage applications across compatible Oracle Linux systems by using Open Container Initiative (OCI) compatible containers.

For information about the Open Container Initiative, visit https://opencontainers.org/.

About Podman

Podman provides a lightweight utility to run and manage Open Container Initiative (OCI) compatible containers. As such, a Podman deployment can reuse existing container images that are designed for Kubernetes, Oracle Container Runtime for Docker, and Oracle Cloud Native Environment.

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

Unlike Oracle Container Runtime for Docker, Podman doesn't require a running daemon to function and has no dependency on the Unbreakable Enterprise Kernel (UEK). Containers run correctly on systems that are running either the Red Hat Compatible Kernel (RHCK) or the UEK release. In addition, Podman containers can start and run without root permissions.

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

The Docker Hub hosts applications as Docker images and provides services that enable you to create and manage compatible containers with Podman. Registering for an account with the Docker Hub enables you to use Podman to store private images. You don't need an account to access publicly accessible images on the Docker Hub. The Docker Hub also hosts enterprise-ready applications that are certified as trusted and supported by their vendors. These applications are made available by the verified publishers. Note that some applications that are shipped on the Docker Hub might require payment.

> **Note:**
>
> The Docker Hub is owned and maintained by Docker, Inc. Oracle makes Docker images available on the Docker Hub that you can download and use with the Docker Engine.
>
> For more information, visit https://docs.docker.com.

The Oracle Container Registry contains 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 non-production use. The Oracle Container Registry provides a web interface where customers can select Oracle images. If required, you must agree to terms of use before pulling the images by using the standard Podman client software. See Using Container Registries for more information about this service.

For general information about Podman, see https://podman.io and the manual pages for Podman.

About Buildah

Buildah is a utility for creating Open Container Initiative (OCI) 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 Building Images With Buildah.

About Skopeo

Skopeo is a utility for managing container images on remote container registries. This utility is useful for inspecting the contents of a container image without needing to first download it.

If you host container images in a container registry, you can use Skopeo to seamlessly move container images from one location to another. In particular, Skopeo is useful for bulk-deleting unneeded container images.

For more information, see Using Skopeo to Inspect and Copy Images.

2

Installing Podman and Related Utilities

The following instructions describe how to install Podman and related tools on an Oracle Linux host. Instructions for removing these tools are also provided.

Podman and its related utilities, Buildah and Skopeo, are designed to work independently of each other. For example, Buildah has no dependency on Podman, which means it's possible to separate the container build infrastructure from environments in which the containers are intended to run. You can install the `buildah` package on the same system that you run Podman; or, you can install the package on another system, if required. Similarly, you can install Skopeo separate from the other utilities according to your specific requirements.

To use Podman, you must have the latest RHCK or UEK version installed.

Podman and related tools are available for Oracle Linux 8 and Oracle Linux 9 on ULN and the Oracle Linux yum server.

To install Podman and related tools, use the following command that matches the system's Oracle Linux OS:

- **Oracle Linux 8**

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

- **Oracle Linux 9**

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

> **Note:**
>
> Initial Application Streams released on Oracle Linux 9 are no longer released in modular format to simplify user experience.

Verifying Podman

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

```
sudo podman info
```

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

You can optionally install the `podman-docker` package that effectively aliases the `docker` command to `podman`. The installed package can help in environments where users are more familiar with Docker or where automation expects the `docker` command to be present.

To install the `podman-docker` package.

```
sudo dnf install podman-docker
```

To remove Podman, stop any running Podman containers and related `systemd` services. For more information, see Managing Containers.

When all containers have been halted or suspended, you can remove the `podman` package.

```
sudo dnf remove podman
```

Verifying Buildah

Check the current version of Buildah by specifying the `--version` flag.

```
sudo buildah --version
```

Use the `buildah -h` command for a command reference.

```
sudo buildah -h
```

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

To remove the `buildah` package.

```
sudo dnf remove buildah
```

Verifying Skopeo

Use the `skopeo -h` command for version information and a command reference.

```
sudo skopeo -h
```

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

To remove the `skopeo` package.

```
sudo dnf remove skopeo
```

3

Working With Images, Containers, and Pods

Podman can be used to run containers and to obtain the images that are used to create a container in the same way that you would use Oracle Container Runtime for Docker. The following information describes how you can pull container images from registries into the local image storage; how you can manage container images on local storage; how you can run containers based on these images; and how you can manage the containers that you have created on the host system.

In many ways, Podman can be used as a drop-in replacement for Oracle Container Runtime for Docker. Podman can use images that comply with the Open Container Initiative (OCI) specification and can run containers based on these images. 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, as it changes the security model around working with images and containers.

Because Podman operations aren't dependent on a service daemon running as a particular user on the system, Podman provides more isolation than Docker. This means that you can either run Podman as a standard user or as the root user.

Podman respects user namespaces. This means that several users on a single host can all run their own containers and local image stores without any concern that there could be a conflict. Because containers running within a user's namespace are limited to the permissions available to the user on the host system, Podman is considered to provide more security than Docker.

Some key differences exist when you run Podman as a standard user, instead of running Podman as the root user. These differences are based on the permissions available to these different user types. For example, networking functionality is more limited when running Podman as a standard user and most networking is achieved solely using port mapping and port forwarding, when in this mode. This doesn't suggest that running Podman as the root user is preferred, but some limitations might apply when working as a standard user. To some degree, many issues that arise for a standard user can be mitigated by running groups of containers within a pod. For more information about networking and Podman, see Configuring Networking for Podman. For more information about pods, see Managing Pods.

In general, the instructions provided here apply similarly regardless of whether Podman runs as a standard user or as the root user.

Running Commands With Podman

You can review a list of the commands available for the installed version of Podman by using the `podman -h` command. For example, that list might contain the following commands:

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

container
Manager 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

generate
Generate systemd unit files and pod YAML files

healthcheck
Run a healthcheck 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

load
Load an image from a container archive file

login
Log in to a container registry

logout
Log out of a container registry

logs
Review logs for a container

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 a Pod

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

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

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

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

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.

Working With Container Images

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 enable you to identify different versions of the same image. Often an image might include a default tag of `latest` so that Podman users can easily identify the most

recent version of the image. Note that Oracle Linux images don't use the `latest` tag. See Oracle Linux Container Image Tagging Conventions for more information.

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 Using Container Registries.

Sometimes, you might want to change an existing image or create custom images, which can be done by using the Buildah utility. See Building Images With Buildah for more information.

Searching for Images in Available Registries

You can search the configured registries for an image by using the `podman search` command:

```
podman search oraclelinux
```

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

Pulling Images From a Registry

When you have found an image, download a copy of it by using the `podman pull` command. When pulling an image, specify the image reference as follows:

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

- The *registry.host* is the resolvable hostname of the registry where the image is hosted. Although the registry host is often required, if the registry is already listed within the podman configuration you don't need to specify this value.
- The *repository* is optional and depends on how images are stored and on the registry.
- The *imagename* is required to specify which image to download.
- The *tag* represents a version of the image and should always be specified. Many tools default to using the `latest` tag if no tag is specified but this can lead to errors and is now considered bad practice. See Oracle Linux Container Image Tagging Conventions for more information on tags and why the `latest` tag is unreliable.

The following example shows how to pull a slim Oracle Linux 7 image from the Oracle Container Registry:

```
podman pull container-registry.oracle.com/os/oraclelinux:7-slim
```

```
Trying to pull container-registry.oracle.com/os/oraclelinux:7-slim...
Getting image source signatures
Copying blob 50ab38810635 done
Copying config d788eca028 done
Writing manifest to image destination
Storing signatures
d788eca028a0f49b6bc70b251c8535b16ee5bd94e0ab375ca8f3a923e6ce4281
```

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

```
podman pull os/oraclelinux:7-slim
```

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

```
podman pull oraclelinux:7-slim
```

The image is downloaded into the local container image store. This storage is described in more detail in Configuring Storage for Podman.

Inspecting an Image

After the download has completed, you can check the default configuration settings and metadata for a container image by running the following command:

```
podman inspect container-registry.oracle.com/os/oraclelinux:7-slim
```

```
[
     {
          "Id":
"6a34bf5396690a3c0ea1d6914f32fae9860bca03c68e7253cfd6d5d11551e2d2",
          "Digest":
"sha256:e63584e0060861cd5301a9fc924b087f32ba28c380126b32ed4874d52bf02051",
          "RepoTags": [
               "container-registry.oracle.com/os/oraclelinux:7-slim"
          ],
          "RepoDigests": [
               "container-registry.oracle.com/os/
oraclelinux@sha256:d7ae060ba190...690de5aa442b8ff8ce46d53895",
               "container-registry.oracle.com/os/
oraclelinux@sha256:e63584e00608...32ba226b32ed4874d52bf02051"
          ],
          "Parent": "",
          "Comment": "",
          "Created": "2022-08-24T19:35:43.028869456Z",
          "Config": {
               "Env": [
                    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/bin"
               ],
               "Cmd": [
                    "/bin/bash"
               ]
          },
          "Version": "20.10.12",
          "Author": "",
          "Architecture": "amd64",
          "Os": "linux",
```

```
          "Size": 139573259,
          "VirtualSize": 139573259,
          "GraphDriver": {
               "Name": "overlay",
               "Data": {
                    "UpperDir": "/home/opc/.local/share/containers/storage/
overlay/bcf2a1ad6472a891de95b...5fb/diff",
                    "WorkDir": "/home/opc/.local/share/containers/storage/
overlay/bcf2a1ad6472a891de95b5...5fb/work"
               }
          },
          "RootFS": {
               "Type": "layers",
               "Layers": [

"sha256:bcf2a1ad6472a891de95b5132d013c064a07ec9995cb61b0cc0f8d4a4ea855fb"
               ]
          },
          "Labels": null,
          "Annotations": {},
          "ManifestType": "application/
vnd.docker.distribution.manifest.v2+json",
          "User": "",
          "History": [
               {
                    "created": "2022-08-24T19:35:42.419825447Z",
                    "created_by": "/bin/sh -c #(nop) ADD
file:adf7ebc1d65494dba22f4f...d6e9964f936343e6d in / "
               },
               {
                    "created": "2022-08-24T19:35:43.028869456Z",
                    "created_by": "/bin/sh -c #(nop)  CMD [\"/bin/bash\"]",
                    "empty_layer": true
               }
          ],
          "NamesHistory": [
               "container-registry.oracle.com/os/oraclelinux:7-slim"
          ]
     }
]
```

Listing Available Images

To list all the locally stored container images that have already been downloaded, type:

```
podman images
```

```
REPOSITORY                                    TAG         IMAGE ID
CREATED     SIZE
container-registry.oracle.com/os/oraclelinux  8-slim      0393d9bbfe64  5
days ago  105 MB
container-registry.oracle.com/os/oraclelinux  7-slim      6a34bf539669  7
days ago  140 MB
```

Deleting an Image

To delete a locally stored container image that have already been downloaded, use the `podman rmi` command.

```
podman rmi oraclelinux:7-slim
```

```
Untagged: container-registry.oracle.com/os/oraclelinux:7-slim
```

> **Note:**
>
> You can't delete an image if it's in use by a container, even if the container isn't running. You must remove all the containers that depend on the image before you can remove the image itself.

Managing 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.

Containers share namespaces and can access shared port mappings to communicate with each other and with the host system. Podman introduces the concept of *pods* to the runtime environment. You can use pods to further isolate a group of containers, thereby making it easier to manage a set of services that work together to provision a logical application. See Managing Pods for more information.

Creating Containers

You can create a container from an existing image using the `podman create` command.

```
podman create -d --name oracle oraclelinux:7-slim
```

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.

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. Run `podman help create` to see more information. Options are extensive and can be used to apply a wide range of runtime functionality to any container.

Running Containers

You run a single command in a container that's provisioned and destroy in a single step by using the `podman run` command with the `--rm` flag, for example:

```
podman run --rm oraclelinux:7-slim cat /etc/oracle-release
```

```
Oracle Linux Server release 7.9
```

You can also 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.

```
podman run --name=oracleshell -it oraclelinux:7-slim /bin/bash
```

```
[root@dcbe94cd0301 /]#
```

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:

```
podman start -ai oracleshell
```

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

```
podman run -d --name=oracledaemon oraclelinux:7-slim /bin/bash -c 'yum
install -y httpd && httpd -Dforeground'
```

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.

> **Note:**
>
> 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. For example:
>
> ```
> podman run --stop-signal SIGHUP --name myol9 oraclelinux:9
> ```
>
> 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
> ```

Enabling FIPS Mode in Containers

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

For more information about Oracle Linux 8 releases that have FIPS validated cryptographic modules available for installation, see Oracle Linux 8: Enhancing System Security. For more information about FIPS on Oracle Linux 9, see Oracle Linux 9: Installing and Configuring FIPS Mode.

> **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.

For Oracle Linux 7 Containers:

To enable FIPS mode in an Oracle Linux 7 container, install the `dracut-fips` package or mount `/etc/system-fips` from the host. For more information about mounting host files and directories inside a Podman container, see Setting Up Container Mounts.

For Oracle Linux 8 and Oracle Linux 9 Containers:

If you have enabled FIPS mode on an Oracle Linux 8 or Oracle Linux 9 host, then Podman runs Oracle Linux 8 or Oracle Linux 9 containers in FIPS mode automatically.

Listing and Monitoring Containers

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

```
podman ps -a
```

```
CONTAINER ID  IMAGE
COMMAND               CREATED         STATUS                     PORTS  NAMES
3a430d30612d  container-registry.oracle.com/os/oraclelinux:7-slim  /bin/bash -
c yum...   20 seconds ago  Up 19 seconds ago                 oracledaemon
dcbe94cd0301  container-registry.oracle.com/os/oraclelinux:7-slim  /bin/
bash             58 seconds ago  Exited (0) 29 seconds ago         oracleshell
726d97e9bfbd  container-registry.oracle.com/os/oraclelinux:7-slim
top                   3 days ago      Created                           oracle
```

To review the logs generated by a container that has already performed actions, use the `podman logs` command.

```
podman logs oracledaemon
```

In the sample command, *oracledaemon* can be either the container's ID or name.

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

```
podman stats oracledaemon
```

```
ID             NAME           CPU %   MEM USAGE / LIMIT   MEM %   NET
IO              BLOCK IO   PIDS
c4e296d0c78e   oracledaemon   4.25%   33.33MB / 29.22GB   0.11%   2.412kB /
1.536kB   -- / --    3
```

Pausing and Resuming Containers

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

```
podman pause oracledaemon
```

Running the previous 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, for example:

```
podman unpause oracledaemon
```

Stopping and Removing Containers

To stop containers by specifying the name or container ID with the `podman stop` command:

```
podman stop oracledaemon
```

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 `-a` flag to the `podman stop` command:

```
podman stop -a
```

Specify the container name or ID with the `podman rm` command to delete a specified container:

```
podman rm oracle
```

Managing 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 Oracle Container Runtime for 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 similar 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.

Finally, by running containers within pods, it's easier 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 Working With Podman Services for more information.

Creating and Managing Pods

The concept of pods is derived from Kubernetes pods. Podman pods are the smallest compute units that you can create and deploy 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.

Create a pod with the `podman pod create` command. Add the `--name` parameter to give the pod a human-readable identifier.

```
podman pod create --name oraclepod
```

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 also be created automatically when a container is run for the first time. See Using Containers Within a Pod for more information.

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

```
podman pod list
```

Remove the pod by using the `rm` command.

```
podman pod rm oraclepod
```

Note that you can only remove a pod when all the containers within the pod have been removed, except for the infrastructure container. By default, an infrastructure 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. You can see which containers are within the pod by using the `podman pod inspect` command.

Using Containers Within a Pod

To attach containers to a pod, use the `--pod` flag when you run the container.

```
podman run -d --pod oraclepod nginx:alpine
podman run --pod oraclepod -it --rm oraclelinux:7-slim curl http://
localhost:80
```

In the previous example, a container using the `nginx` image is run and connected to the pod named *oraclepod*. A second container, using the `oraclelinux` image is started and connected to the same pod. The `curl` command is run in the second container to access the web service running on `localhost` on port 80. The containers are both running as standard user 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.

If a pod doesn't already exist, you can create it directly by using the `podman run` command with the `--pod` option and prepending the `new:` option to the human-readable name that you have chosen as the pod name:

```
podman run -d --pod new:oraclepod nginx:alpine
```

Review all the containers on a host with the pod to which the containers are attached by including the `--pod` or `-p` flag with the `podman ps -a` command.

```
podman ps -ap
```

You can start and stop containers as usual without affecting the entire pod; however, you can also use the `podman pod start` and `podman pod stop` commands to start and stop every container that exists within the same pod simultaneously, for example:

```
podman pod stop oraclepod
podman pod start oraclepod
```

To check the current status of a pod, use the `podman pod ps` command:

```
podman pod ps
```

4

Configuring Storage for Podman

The following information describes how to add and configure storage for Podman and related utilities.

By default, images are stored in the `/var/lib/containers` directory when Podman is run by the root user. For standard users, images are typically stored in `$HOME/.local/share/containers/storage/`. These locations conform to Open Container Initiative (OCI) 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 locations can be set up as mount points to take advantage of some forms of network storage or of dedicated local file systems, depending on requirements.

> **Caution:**
>
> When containers are run by users without root permissions, Podman lacks 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 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 storage configuration is handled by editing the file at `/etc/containers/storage.conf`.

You can override the system-wide storage configuration by creating a separate `$HOME/.config/containers/storage.conf` configuration file for any user.

The following is an example of a typical storage configuration file:

```
cat ~/.config/containers/storage.conf


[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 might help you better understand the information in this 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 usually 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 is unshared 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.

5

Configuring Networking for Podman

The networking information in this chapter enable you to understand how to configure the different networking requirements for containers that run within 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 within 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 Creating and Managing Pods.

Unless indicated otherwise, all the networking procedures in the sections that follow can be performed only by the root or privileged user.

Configuring Proxy Server Settings

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. 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 `/etc/systemd/system/podman.service.d/http-proxy.conf` 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
```

> **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.

Configuring Port Mapping for Containers

Defining port mappings can be performed by both the root user and the standard user.

For containers that are run by the standard user, Podman relies on port mapping to use the existing network infrastructure that's available on the host system. Thus, a standard user can't and doesn't need to configure specific network settings such as assigning IP addresses for such containers. Podman handles the networking functionality for these containers automatically by performing port forwarding to container-based services.

Port publishing for a non-root user is limited to port numbers that are outside the range of the privileged port numbers, whose limit is port 1024. Therefore, when creating containers that are using a specific port mapping, ensure that the port number is set to a value greater than 1024. For example, to map port 8080 on the host to the container port 80, type:

```
podman run -d -p 8080:80/tcp nginx:alpine
```

You can also use the `-P` option 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, you can access the port directly from the host where the container is running. From the previous example, the host can access port 80 on the container by opening a web browser to `http://localhost:8080`.

You can view a container's port mappings directly by using the following command:

```
podman port container_id
```

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

You can also see port mappings when you inspect a container. Use the `podman port -a` command to view all port mappings for all the containers running on the host.

Because the containers and the host share the same network namespace, a container can communicate directly with another container by using the IP address and the port mapping that the parent host uses. The easiest way for one container to connect to a port on another container is to connect to the host IP address and port mapping.

Extending the previous example, you can run a second container by using the following command, where 198.51.100.10 is the IP address of the host system:

```
podman run -it --rm oraclelinux curl http://198.51.100.10:8080
```

External hosts can also connect to the container port mapping by using the host's IP address and the mapped port. However, if the host has firewall software running, you might need to open the firewall port for the container to be externally accessible.

Regardless of whether port mapping is performed by the root user or a standard user, the functionality operates in the same way for their respective containers.

Networking for containers that are run by standard users is understandably limited. Similar constraints apply to users who are setting up applications on a host within userspace. For more complex networking, containers must be run as the root user.

Configuring 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. See https://github.com/containernetworking/cni for more information.
- Netavark: A network stack designed specifically for Podman, but compatible with other OCI container management applications. See https://github.com/containers/netavark

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:

```
sudo podman info | grep networkBackend
```

> **NOT_SUPPORTED:**
>
> 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 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 broadly used 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 install the `containernetworking-plugins` package.

For each network that you create within 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, as Podman can handle the files on your behalf.

A typical network configuration file might appear as follows:

```
{
   "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, particularly around 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 install the `netavark` package.

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, as Podman can handle the files on your behalf.

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 inspect network_name
```

Changing 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:

```
sudo podman network create network_name
```

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 to obtain more information about these options.

To remove a network that you have created, run:

```
sudo podman network rm network_name
```

> **Note:**
>
> To remove a network, you must first remove any container that's connected to the network. You can use the `-f` option to force the removal of any containers that are using the network.

You can remove all networks that are unused on the system by typing:

```
sudo podman network prune
```

Listing Networks

List all the Podman networks you have created:

```
sudo podman network ls
```

```
NETWORK ID    NAME        DRIVER
2f259bab93aa  podman      bridge
47d141b0df17  podman1     bridge,portmap,firewall,tuning
```

Connecting and Disconnecting Container Networks

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.

If you have created a new network, you can add any container to the network by running:

```
sudo podman network connect network_name container_id
```

You can disconnect a container from a network by running:

```
sudo podman network disconnect network_name container_id
```

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_id
```

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

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

To view port mappings for a container, run:

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

6
Working With Podman Services

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.

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 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.

Setting SELinux Permissions for Container and Pod 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
```

About Quadlets

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 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 whenever 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.

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

Creating Quadlet Containers

Quadlet containers are managed by using a unit file with the `.container` file extension. It must also contain a `[Container]` section, for example:

```
[Unit]
Description=your-description
```

```
After=local-fs.target

[Container]
Image=container-registry.oracle.com/os/oraclelinux:8-slim
Exec=your-command-here

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

To create a Quadlet container that runs with root permissions, store that `.container` file in either of the `/usr/share/containers/systemd/` or `/etc/containers/systemd/` directories, and then reload Systemd services:

```
sudo systemctl daemon-reload
```

To create a Quadlet container that runs with standard user permissions, administrators can create quadlet files in the `/etc/containers/systemd/users/${USER_ID}` directory for each individual user, or individual users can store `.container` files in their' `~/.config/containers/systemd/` directory, and then reload Systemd services:

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

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.

You can now manage the Quadlet the same way as any other Systemd service by using the `systemctl` command. For example, to start a quadlet container called `myoracle` that was confirmed during the dry run, run the following command:

```
sudo systemctl start myoracle
```

For more information, see the `podman-systemd.unit(5)` manual pages.

Using the Podman Login Shell

In Oracle Linux 9, 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 command 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 About Quadlets.

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 `${USER_ID}`, and the quadlet is `podmansh.container`.

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

[Container]
Image=container-registry.oracle.com/os/oraclelinux:8-slim
ContainerName=podmansh
RemapUsers=keep-id
RunInit=yes

Exec=sleep infinity

[Install]
RequiredBy=default.target
```

You can define more parameters in the [`Container`] section to further specify what a user can do when inside a container. See the `podman-systemd.unit(5)` manual page for details.

Note that the name of the container (`ContainerName`) must be `podman`. In this manner, the `/usr/bin/podmansh` shell command runs `podman exec` in the `podmansh` container.

The user with the ID that matches the quadlet's `${USER_ID}` would log in as follows:

```
ssh user@systemname
```

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.

Generating Podman Service Wrappers

Instead of writing a Systemd service wrapper from scratch, 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/`.

Generating Podman Service Wrappers for Containers

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
```

Generating Podman Service Wrappers for Pods

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.

Managing Podman Services

Systemd services are all managed by using the `systemctl` command.

After you have configured Systemd service wrappers for any containers or pods, you can use `systemctl` commands to manage those containers or pods as services.

If you're running containers as a standard user, all `systemctl` commands must use the `--user` option.

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
```

For more information, see https://docs.oracle.com/en/learn/use_systemd/index.html .

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 8 by following the instructions at Oracle Linux 8: Managing Core System Configuration. On Oracle Linux 9, see Oracle Linux 9: Managing Core System Configuration for more information.

For more information about how you can make modifications to the service wrapper you have generated with the `podman generate systemd` command, see https://docs.podman.io/en/latest/markdown/podman-generate-systemd.1.html.

7

Building Images With Buildah

This chapter describes how to use Buildah to create new images for use with Podman.

The Buildah utility is functionally similar to Podman in the way that it behaves, but maintains independence from Podman to facilitate the build of OCI compliant images. The primary difference between Buildah and Podman is in the way that the `run` command is handled. Because Buildah's purpose 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 required 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` that works with Oracle Container Runtime for Docker 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 that 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.

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

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

Creating Images From Containerfiles With Buildah

Buildah, which is designed to work directly with an existing `Containerfile`, processes the file to build an image using the 'build using dockerfile' or `bud` command. You can use any `Containerfile` that works with Oracle Container Runtime for Docker to build an image and the `buildah bud` command behaves similarly to the `docker build` command.

To build an image from a `Containerfile`, navigate to the directory where the file is located and type:

```
sudo buildah bud -t imagename .
```

Use the `-f filename` option if the file doesn't use the default `Containerfile` file name.

The built image is then tagged and added to the local image list. To confirm that the new image is available, type:

```
sudo buildah images
```

Note that because locally hosted images are shared between Buildah and Podman, the image is immediately available for use within Podman. However you might consider pushing the images to a registry where these images can become available to other systems where Podman is installed.

To verify an image by running it within a working container, type:

```
sudo buildah from imagename
```

The `buildah from` command creates a working container for any image. For more details, see Changing Images With Buildah.

Changing Images With Buildah

You change images by running them as working container instances within Buildah and issuing `buildah run` commands. These commands are the same as `RUN` statements in a `Containerfile`. After completing the changes to the working container, run the `buildah commit` to generate a new image based on the current status of the working container.

Setting Up Images and Working Containers

Buildah can pull images from a registry in the same way that you would pull an image by using Podman. For example, to pull the `oraclelinux:7-slim` image, you would type:

```
sudo buildah pull container-registry.oracle.com/os/oraclelinux:7-slim
```

Locally hosted images are shared between Buildah and Podman. To start a working container from any image within Buildah, use the `buildah from` command, for example:

```
sudo buildah from oraclelinux:7-slim


oraclelinux-working-container-1
```

You can combine a pull request with the `buildah from` command to start a container from an image hosted on a remote registry:

```
sudo buildah from container-registry.oracle.com/os/oraclelinux:7-slim
```

You can create and start a working container in any of the following ways:

- Using a local image that you have pulled by using either Podman or Buildah
- Using an image that was generated from a `Containerfile`
- Using a remote image that you pulled directly from a remote registry.

To check which containers are available within Buildah, issue the following command:

```
sudo buildah containers
```

Making Changes to the Working Container

One of the key differences between Podman and Buildah is the handling of the `run` command. With Podman, this command emulates the equivalent Docker command and runs a process in a new container. With Buildah, the command behaves the same as a `RUN` statement from a `Containerfile` and affects the existing working container. You can use this command to run a series of commands against the working container to validate it, and to change it to a point where you're ready to store it as an image.

Use the `buildah run` command to run shell commands in the working container, such as installing packages and make system changes. The following command displays the Oracle Linux release in a specific container:

```
sudo buildah run oraclelinux-working-container-1 cat /etc/oracle-release


Oracle Linux Server release 8.2
```

To copy files to the working container, type:

```
sudo buildah copy oraclelinux-working-container-1 /path/to/yourscript /usr/
local/bin
```

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

```
sudo buildah mount oraclelinux-working-container-1


/var/lib/containers/storage/overlay/5a595715c70a2d1a5582836d55722fbfc2ab9bf3/
merged
```

The command returns the mount point where you can access the root file system for the container.

To change the file system to reflect directly within the container, type:

```
sudo touch /var/lib/containers/storage/overlay/
5a595715c70a2d1a5582836d55722fbfc2ab9bf3/merged/testfile
sudo buildah run oraclelinux-working-container-1 ls /testfile


testfile
```

If you run the `buildah mount` command without any parameters, all mounts are displayed so that you can see which containers are mounted and the path to the mount point on the host system. Containers are listed by container ID, so you would need to match these to countainer names based on the information returned by the `buildah containers` command.

As a best practice, unmount container file systems when you're finished working on them by using the `buildah unmount` command. Note that this command isn't aware of container names, so you must use the command with the container ID, for example:

```
sudo buildah unmount 5617ebfbe265
```

Creating a New Image From the Working Container

Changes to the working container are temporary until they're committed to an image. Buildah enables you to commit an image locally for immediate use by Podman. However, Buildah also provides the facility to commit to a registry where you have write access. A range of options are available to help handle registry related access, such as authentication and certificate validation requirements. To find out about these options, see the `buildah-commit(1)` manual page.

To permanently store the changes to a working container as an image, use the `buildah commit` command. By running this command, you can start similar containers within the Podman environment. In the following example, the working container is committed locally and the `--rm` option is used to remove the working container from the Buildah environment at the same time:

```
sudo buildah commit --rm oraclelinux-working-container-1 new-image
```

To commit to a local registry, you would type:

```
sudo buildah commit --rm oraclelinux-working-container-1 docker://
localhost:5000/new-image
```

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

You can also review the build image metadata with the `buildah inspect` command. The command output can help identify historical changes and other information about the image or working container, for example:

```
sudo buildah inspect --type image new-image
```

Removing a Working Container

The `buildah rm` command stops and removes a container from Buildah's container list.

```
sudo buildah rm oraclelinux-working-container-1
```

This command removes the working container and unmounts any mounts that might have been set up for it. After the container is removed, any changes that you made to the working container are lost and unretrievable unless you created an image of the working container before you removed it.

This command only removes the working container from the Buildah environment. Any containers running under Podman nor any existing images are unaffected. Because images are shared between Podman and Buildah, if you use the `buildah rmi` command to remove an image, the image is also removed for Podman.

> **Note:**
>
> As described in previous sections, using the `--rm` option with the `buildah commit` command enables you to both commit a working container and remove it from Buildah's container list at the same time. See Creating a New Image From the Working Container.

Pushing Images to a Registry

Buildah and Podman handle images interchangeably, so most of the commands that you run 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 that exists in local storage to a registry where you have write access.

To push a local image to a registry that uses the Docker Registry HTTP API V2, run the following command:

```
sudo buildah push imagename docker://registry.example.com/imagename:tag
```

Because Podman tries to maintain syntax compatibility with the Docker CLI, you can run this command by using the following syntax. The following sample command assumes that the local image is stored with its registry name and tag.:

```
sudo buildah push registry.example.com/imagename:tag
```

Registries are HTTP based resources that are typically protected by using TLS/SSL certificates and might require authentication. Several options that are related to how the command handles these requirements are available. For more information about these options, see the `buildah-push(1)` manual page.

Buildah and Podman are equally able to push images to other formats. Use this feature to 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, similar to the results of running the `docker save` command in that environment. Type:

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

Substitute `docker-archive` in the command with `oci-archive` to generate an archive that complies with the Open Container Initiative (OCI) specification.

8

Using Skopeo to Inspect and Copy Images

This chapter explains how to use Skopeo to inspect and copy 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 OCI-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 in a Remote Registry

Use the `skopeo inspect` command to inspect information about an image within a registry, 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 alternative available tags for a matching image name in the registry. For example, to display information about the `oraclelinux:8-slim` image, you would type:

```
skopeo inspect docker://docker.io/library/oraclelinux:8-slim
```

```
{
    "Name": "docker.io/library/oraclelinux",
    "Digest":
"sha256:410ddd2c0df85b96dc43af11530df8a7adc554e2a61b2645ff3893e53a6e6813",
    "RepoTags": [
        "5.11",
        "5",
        "6-slim",
        "6.10",
        "6.6",
        "6.7",
        "6.8",
        "6.9",
        "6",
        "7-slim",
        "7.0",
        "7.1",
        "7.2",
        "7.3",
```

```
            "7.4",
            "7.5",
            "7.6",
            "7.7",
            "7.8",
            "7",
            "8-slim",
            "8.0",
            "8.1",
            "8.2",
            "8"
        ],
        "Created": "2020-06-02T16:25:27.035497362Z",
        "DockerVersion": "18.09.7",
        "Labels": null,
        "Architecture": "amd64",
        "Os": "linux",
        "Layers": [

"sha256:962e54b445ab56928c06f1d40aebe99a80b10b2bc428260bc931b24cec46e11c"
        ],
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
        ]
}
```

To review the image's default configuration settings and build history, use the `--config` option.

```
skopeo inspect --config docker://docker.io/library/oraclelinux:8-slim
```

```
{
    "created": "2020-06-02T16:25:27.035497362Z",
    "author": "Oracle Linux Product Team <ol-ovm-info_ww@oracle.com>",
    "architecture": "amd64",
    "os": "linux",
    "config": {
        "Env": [
            "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin"
        ],
        "Cmd": [
            "/bin/bash"
        ]
    },
    "rootfs": {
        "type": "layers",
        "diff_ids": [

"sha256:4beda459d2bfe9960c5537dc4e04b43ffdc8f409897e092df2c0cc2094d82d31"
        ]
    },
    "history": [
        {
            "created": "2018-08-30T21:49:27.028879762Z",
```

```
            "created_by": "/bin/sh -c #(nop)  MAINTAINER Oracle Linux Product
Team <ol-ovm-in...",
            "author": "Oracle Linux Product Team <ol-ovm-info_ww@oracle.com>",
            "empty_layer": true
        },
        {
            "created": "2020-06-02T16:25:26.6629159Z",
            "created_by": "/bin/sh -c #(nop) ADD file:336b... in / ",
            "author": "Oracle Linux Product Team <ol-ovm-info_ww@oracle.com>"
        },
        {
            "created": "2020-06-02T16:25:27.035497362Z",
            "created_by": "/bin/sh -c #(nop)  CMD [\"/bin/bash\"]",
            "author": "Oracle Linux Product Team <ol-ovm-info_ww@oracle.com>",
            "empty_layer": true
        }
    ]
}
```

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

Copying an Image Between Container Storage Types

Use the `skopeo copy` command to copy an image between registries without needing to download it locally first.

```
skopeo copy docker://docker.io/library/oraclelinux:8-slim docker://
example.com/os/oraclelinux:8-slim
```

If the destination registry requires a signature, provide the required `key-id` by using the `--sign-by` parameter.

You can also copy an image to local Podman container storage by adding the `containers-storage:` prefix to the image name.

```
skopeo copy docker://docker.io/library/oraclelinux:8-slim containers-
storage:oraclelinux:8-slim
```

To download an image and review its internal content offline, specify a directory with the `dir:` prefix. For example, to extract the to the `oraclelinux:8-slim` image to the `oraclelinux` folder in a home directory, you would type:

```
skopeo copy docker://docker.io/library/oraclelinux:8-slim dir:/home/$USER/
oraclelinux
```

In the example, the `oraclelinux` folder contains a `manifest.json` file and multiple tarballs representing the image that has been copied.

For more information about the prefixes and flags you can use, see the `skopeo-copy(1)` manual page.

Deleting an Image From Container Storage

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

```
skopeo delete containers-storage:oraclelinux:8-slim
```

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

9
Using Container Registries

This chapter describes how to sign in to Oracle Container Registry, create a self-hosted container registry, and add new container 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 three commonly used registries:

- **container-registry.oracle.com**

  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. Users can pull container images with the familiar Docker Command Line Interface (CLI) and API.

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

  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.

  To use licensed Oracle software images, first sign in to the Oracle Container Registry web interface and accept the Oracle Standard Terms and Restrictions for the required software images.

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

- **quay.io**

  The Quay Container Registry is a popular registry provided by Red Hat. Many open source images are available at this registry.

- **docker.io**

  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 at https://hub.docker.com.

  Enterprise-ready images from Oracle are also available on the Docker Hub.

Enterprise environments might also consider setting up a local container registry. The local container registry would store images that were 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.

Registry Configuration

To configure default registry settings, edit `/etc/containers/registries.conf`.

The configuration file is commented to explain the options that are available. 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:

```
[registries.search]
registries = ['container-registry.oracle.com', 'quay.io', '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.

To use an insecure registry without a valid SSL certificate or that doesn't use SSL, add the registry domain name to the `registries` list in the `[registries.insecure]` configuration block.

Configuring Podman for Signed Images

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. For each registry where you require signature validation, create a YAML format configuration file in `/etc/containers/registries.d/` and provide the value for the `sigstore` for that registry.

   For example, for the Oracle Container Registry, create a file `/etc/containers/registries.d/oracle.yaml` 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 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 at 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. 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 that the configuration is correct by pulling an image from the remote registry. See Pulling Images From the Oracle Container Registry for an example. Note that if the signature requirement is configured correctly, the output isn't different from an output when you pull an image without signature validation configured. 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`. Signature validation failure appears as follows:

```
Trying to pull container-registry.oracle.com/os/oraclelinux:7-slim...
 Invalid GPG signature: gpgme.Signature{Summary:128,
Fingerprint:"357217938FC350A2",
```

```
 Status:gpgme.Error{err:0x9}, Timestamp:time.Time{wall:0x0,
ext:63754125715,
 loc:(*time.Location)(0x558c35f0c0a0)}, ExpTimestamp:time.Time{wall:0x0,
ext:62135596800,
 loc:(*time.Location)(0x558c35f0c0a0)}, WrongKeyUsage:false, PKATrust:0x0,
 ChainModel:false, Validity:0, ValidityReason:error(nil), PubkeyAlgo:1,
HashAlgo:2}
Error: Source image rejected: Invalid GPG signature:
gpgme.Signature{Summary:128,
 Fingerprint:"357217938FC350A2", Status:gpgme.Error{err:0x9},
Timestamp:time.Time{wall:0x0,
 ext:63754125715, loc:(*time.Location)(0x558c35f0c0a0)},
ExpTimestamp:time.Time{wall:0x0,
 ext:62135596800, loc:(*time.Location)(0x558c35f0c0a0)},
WrongKeyUsage:false, PKATrust:0x0,
 ChainModel:false, Validity:0, ValidityReason:error(nil), PubkeyAlgo:1,
HashAlgo:2}
```

Pulling Images From the Oracle Container Registry

If you're pulling a licensed Oracle software image, you must first sign in to the Oracle Container Registry and accept the Oracle Standard Terms and Restrictions. For information about pulling licensed Oracle software from the Oracle Container Registry, see Pulling Licensed Software From the Oracle Container Registry.

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

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

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

```
sudo podman pull container-registry.oracle.com/os/oraclelinux:7-slim
```

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

> **Note:**
>
> The correct command to pull an image is usually 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.

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. You can create an Oracle Account at https://profile.oracle.com/myprofile/account/create-account.jspx.

> **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.

Pull a licensed software image from the Oracle Container Registry as follows:

1. 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 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. Browse or search for Oracle software images.
4. On the host system, use the `podman login` command to authenticate against the Oracle Container Registry.

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

   Provide an Oracle account username and password as prompted.
5. Pull the images that you require by using the `podman pull` command, for example:

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

   For more detailed information about pulling images from the Oracle Container Registry, see Pulling Images From the Oracle Container Registry.

   The image is pulled from the Oracle Container Registry and stored locally, ready to be used to deploy containers.
6. 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:

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

Using the Oracle Container Registry Mirrors With Podman

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 by using the Oracle Container Registry web interface. The `Tags` table at the bottom of 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.

For example, the command to pull the Oracle Linux 7 image from the Sydney mirror would be the following:

```
sudo podman pull container-registry-sydney.oracle.com/os/oraclelinux:7-slim
```

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, which is at https://container-registry.oracle.com .

To pull licensed Oracle software images, sign in to the Oracle Container Registry mirror before you pull the image, for example:

```
sudo podman login container-registry-sydney.oracle.com
sudo podman pull container-registry-sydney.oracle.com/java/serverjre
sudo podman logout container-registry-sydney.oracle.com.oracle.com
```

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 Registry Configuration for more information.

Using the Docker Hub With Podman

The Docker Hub contains Docker images for licensed commercial Oracle software products that you might use in your enterprise. The Docker Hub is at https://hub.docker.com.

You can browse the Docker Hub and to pull some images from the hub anonymously. However, to access most images hosted in the Docker Hub, you must sign in with a valid Docker ID. You can register for a Docker ID at https://hub.docker.com/signup.

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 click `Proceed to Checkout` button 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.

The following example illustrates how you can sign in to the Docker Hub, inspect, and pull an image:

```
sudo podman login
sudo skopeo inspect docker://docker.io/store/oracle/database-
enterprise:12.2.0.1
sudo podman pull docker.io/store/oracle/database-enterprise:12.2.0.1-slim
```

Setting Up a Local Container Registry

This section contains information about setting up a local container registry server, which can be used to host images, and can also be used as a mirror for the Oracle Container Registry.

The registry server is a container application. The host must have an Internet connection to download the registry image either from the Docker Hub or, if support is required, from the Oracle Container Registry.

The registry server requires at least 15 GB of available disk space to store registry data, typically in `/var/lib/registry`, although you can select another path if you intend to run the registry as a standard user. As a good practice, create a separate file system for the local container 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.

Setting Up Transport Layer Security for the Registry

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. This section discusses adding the host's X.509 certificate and private key to Podman.

If the host already has an X.509 certificate, you can use the same certificate with Podman.

If the host doesn't have an X.509 certificate, you can create a self-signed, private certificate for testing purposes. For information about creating a self-signed certificate and private key, see Oracle Linux: Managing Certificates and Public Key Infrastructure.

To disable X.509 certificate validation for testing purposes, see Registry Configuration.

Use the X.509 Certificate with Podman as follows:

1. If the host's X.509 certificate was issued by an intermediate Certificate Authority (CA), combine the host's certificate with the intermediate CA's certificate to create a chained certificate to enable Docker to verify the host's X.509 certificate, for example:

   ```
   sudo cat registry.example.com.crt intermediate-ca.pem > domain.crt
   ```

2. Create the `/var/lib/registry/conf.d` directory to store the certificate and private key.

   ```
   sudo mkdir -p /var/lib/registry/conf.d
   ```

3. Copy the certificate and private key to the `/var/lib/registry/conf.d` directory.

   ```
   sudo cp certfile /var/lib/registry/conf.d/domain.crt
   sudo cp keyfile /var/lib/registry/conf.d/domain.key
   ```

   In the command, *certfile* is the full path to the host's X.509 certificate or to the chained certificate and *keyfile* is the full path to the host's private key, for example:

   ```
   sudo cp /etc/pki/tls/certs/registry.example.com.crt /var/lib/registry/
   conf.d/domain.crt
   sudo cp /etc/pki/tls/private/registry.example.com.key /var/lib/registry/
   conf.d/domain.key
   ```

4. Ensure the file permissions are correct for the private key.

   ```
   sudo chmod 600 /var/lib/registry/conf.d/domain.key
   ```

Creating the Registry

This section discusses creating the registry server as a container application. Perform these steps on the registry host.

Create the Podman registry container. For example:

```
sudo podman run -d -p 5000:5000 --name registry --restart=always \
    -v /var/lib/registry:/registry_data \
    -e REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY=/registry_data \
    -e REGISTRY_HTTP_TLS_KEY=/registry_data/conf.d/domain.key \
    -e REGISTRY_HTTP_TLS_CERTIFICATE=/registry_data/conf.d/domain.crt \
    -e REGISTRY_AUTH="" \
    container-registry.oracle.com/os/registry:v2.7.1.1
```

The registry image is pulled from the Oracle Container Registry and the registry container is started.

The `--restart=always` option starts the registry container when it's started.

By default, the port number used by container services is 5000. If needed, you can use a different port number for a container registry and map that port to 5000, for example *portnumber*:5000. For more information about mapping ports in Podman, see Configuring Networking for Podman.

If you don't have an Oracle account and don't require support, you can alternately use the publicly available container registry image at `docker.io/library/registry:latest`.

Setting Up the Registry Port

The registry server runs on port 5000 by default. If you run other services that use the same TCP port, such as the OpenStack Keystone service, you might need to change the configuration to avoid a port conflict. All systems that require access to the registry server must be able to communicate on this port. Ensure that you adjust any firewall rules to prevent port conflict..

If you're running a firewall, ensure the TCP port that you want the container registry to listen on is accessible. If you're running `firewalld`, add the default rule for the `docker-registry` service.

```
sudo firewall-cmd --zone=public --permanent --add-service=docker-registry
```

If you don't run the registry on the default port, you can specify the port directly, for example:

```
sudo firewall-cmd --zone=public --permanent --add-port=5001/tcp
```

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, certifcates can be stored in `$HOME/.local/share/containers/certs.d/`*`registry_hostname`*`:`*`port`*`/`.

Podman, Buildah, and Skopeo commands that interact with registries also usually support a `--cert-dir` option to specify an alternate location for these certificates.

Perform the following steps **on each host** that needs to access the local registry. Substitute *registry_hostname* with the name of the registry host, and *port* with the port number you selected for your container registry server (5000 by default).

To distribute a self signed X.509 certificate:

1. Create the appropriate `certs.d` location for the registry host and your user. For example, for the root user, create a directory at `/etc/containers/certs.d/`*`registry_hostname`*`:`*`port`*.

   ```
   sudo mkdir -p /etc/containers/certs.d/registry_hostname:port
   ```

2. Copy the X.509 certificate from the registry host.

   ```
   sudo scp root@registry_hostname:/var/lib/registry/conf.d/domain.crt \
   /etc/containers/certs.d/registry_hostname:port/ca.crt
   ```

Importing Images Into a Registry

When you have set up a container registry server, 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, to pull an image from the Oracle Container Registry, type:

   ```
   sudo podman pull container-registry.oracle.com/os/oraclelinux:7-slim
   ```

2. Tag the image so that it points to the local registry. For example:

   ```
   sudo podman tag container-registry.oracle.com/os/oraclelinux:7-slim
   localhost:portnumber/ol7image:v1
   ```

   In this example, *localhost* is the hostname of the local registry, and *portnumber*, which, by default, is 5000, is the port number that the registry listens on. If you're working on a Podman installation on a different host to the registry, you must change the hostname to point to the correct host. Note the repository and tag name, *ol7image*:*v1* in the example, must all be in lowercase to be a valid tag.

3. Push the image to the local registry, for example:

   ```
   sudo podman push localhost:portnumber/ol7image:v1
   ```

See Building Images With 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.

10

Security Recommendations

Ensure that infrastructure and containerized applications remain secure by following security recommendations and guidelines.

Best Practices for Podman Components

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.

Note that 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 your 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 operating system software**

  Security patches and bug fixes for the kernel and operating system software are released as issues are resolved. Keep the operating system current with the most recent software updates. Subscribe the system to the latest software channels or repositories; run regular DNF update operations; and consider using Ksplice to keep the system software up-to-date.

- **Use a minimal operating system and ensure that it's following security best practices**

  Where possible, use a minimal operating system installation and ensure that it follows the security best practices described in Oracle Linux 8: 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 breakout.

- **Regularly scrutinize the operating system and kernel for safety**

  Be vigilant that the operating system 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 that 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 multiple 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, Podman images should come from and be 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:8
  ```

  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 potentially 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.

11

Known Issues

This chapter describes known issues in the current release of Podman.

Quadlets Fail on Oracle Linux 8 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.

Oracle Linux 9 hosts aren't affected by this issue.

(Bug 36250501)

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)

Container Storage is Not Accessible to an Unprivileged User

Container storage might fail to mount for an unprivileged user in some 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 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 your 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

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:7-
  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:7-
  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 When Attempting to Pull an Image By Specifying 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:ol7-slim` was specified as the name of the image instead of `oraclelinux:ol7-slim`, which is the correct name for the image:

```
podman pull oracle:ol7-slim


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:ol7-slim": unable to pull oracle:ol7-slim: 3
errors
occurred:
```

```
* Error determining manifest MIME type for
docker://registry.redhat.io/oracle:ol7-slim: unable to retrieve auth token:
invalid username/password
* Error determining manifest MIME type for docker://quay.io/oracle:ol7-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:ol7-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

Attempts to search for or pull the `oraclelinux` image from Docker Hub fail with an "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:7` 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)

12

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 just enough operating system 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 7 slim image, use the `7-slim` tag. To use the most recent version of an Oracle Linux 8 slim image, use the `8-slim` tag.

```
sudo podman pull oraclelinux:7-slim
```

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.

The following `slim-fips` images are available:

- `oraclelinux:7-slim-fips`:
  - The latest FIPS compliant versions Oracle Linux 7 cryptographic packages at the time of the release of the image are already installed;
  - The Oracle Linux 7.8 security validation repository is already enabled in the image yum configuration file, so that the container can retrieve system updates that include FIPS compliant cryptographic package versions;
  - The `dracut-fips` package required for container FIPS mode is already installed.
- `oraclelinux:8-slim-fips`:
  - The latest FIPS compliant versions Oracle Linux 8 cryptographic packages at the time of the release of the image are already installed;
  - The Oracle Linux 8.4 security validation repository is already enabled in the image yum configuration file, so that the container can retrieve system updates that include FIPS compliant cryptographic package versions;
  - The `/etc/system-fips` file required for container FIPS mode in docker is already created.

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 operating system than the version provided in a slim image, use a release tag to obtain the latest image for that Oracle Linux image.

For example, to get the latest update release image for Oracle Linux 8, use the `8` tag:

```
sudo podman pull oraclelinux:8
```

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 of 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 8.2 is indicated using the `8.2` tag:

```
sudo podman pull oraclelinux:8.2
```

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 regularly breaks 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:

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

```
Trying to pull docker.io/library/oraclelinux...
  manifest unknown: manifest unknown
Error: error pulling image "docker.io/library/oraclelinux": unable to pull
docker.io/library/oraclelinux:
unable to pull image: Error initializing source docker://oraclelinux:latest:
Error reading manifest latest
in docker.io/library/oraclelinux: manifest unknown: manifest unknown
```

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

```
sudo podman pull oraclelinux:8
```