Oracle® Linux

Podman User's Guide

F30921-10
September 2021

Abstract

Oracle® Linux: Podman User's Guide describes how to create and maintain containers, pods, and images with Podman, Buildah and Skopeo technologies.

Document generated on: 2021-09-08 (revision: 12427)

Table of 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.

Document generated on: 2021-09-08 (revision: 12427)

Audience

This document is intended for administrators who need to install, configure and use the Podman on Oracle Linux 8. It is assumed that readers are familiar with web and virtualization technologies and have a general understanding of the Linux operating system.

Related Documents

The documentation for this product is available at:

Oracle® Linux Documentation

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.

Chapter 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 re-use 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 command-line interface (CLI) functions the same way if the `podman-docker` package is installed.

Unlike Oracle Container Runtime for Docker, Podman does not require a running daemon to function. Also, there is 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 enables containers to start and run without root permissions.

Much like 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 your own private images. You do not 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. These applications are made available by the verified publishers. Note that some applications that are shipped on the Docker Hub may 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 are able to select Oracle images. If required, you must agree to terms of use before pulling the images by using the standard Podman client software. See Chapter 9, *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 Intiative (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 do not need a running daemon for the utility to function. Buildah also does not cache builds by default. In addition, the utility can push container images to container registries, so it is well-suited for use with deployment scripts and automated build pipelines.

For more information, see Chapter 7, *Building Images With Buildah*.

About Skopeo

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

If you host container images in your own 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 Chapter 8, *Using Skopeo to Inspect and Copy Images*.

Chapter 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.

As much as possible, 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 is 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 an alternate 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 on ULN and the Oracle Linux yum server. You can install all of these packages by installing the `container-tools` module using the `dnf` command:

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

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.

For convenience, you may optionally install the `podman-docker` package that effectively aliases the `docker` command to `podman`. This 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 currently running Podman containers and related `systemd` services. For more information, see Section 3.3, “Managing Containers”.

When all your containers have been halted or suspended, you can safely 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
```

Chapter 3 Working With Images, Containers, and Pods

Table of Contents

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. The majority of 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 significantly.

Because Podman operations are not dependent on a service daemon running as a particular user on the system, Podman provides more isolation than Docker. This means that it is equally as straightforward to run Podman as a standard user as it is to run Podman as the root user.

Podman respects user namespaces. This means that multiple 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 generally considered to provide more security than Docker.

There are some key differences between running Podman as a standard user or running Podman as the root user. These differences are naturally based on the permissions available to these different user types. For example, networking functionality is significantly 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 does not suggest that running Podman as the root user is preferred, but you should be aware that some limitations may apply when working as a standard user. To some degree, many issues that arise for a standard user may be mitigated by running groups of containers within a pod. For more information about networking and Podman, see Chapter 5, *Configuring Networking for Podman*. For more information about pods, see Section 3.4, “Managing Pods”.

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

3.1 Running Commands with Podman

You can review a list of the commands available for your installed version of Podman by using the `podman -h` command. For example, that list may 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 a new 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 is 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	Display 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 your 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 an additional 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 an additional name from an image in local storage on the host system
version	Display 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.

3.2 Working With Container Images

A container image is a read-only template that is used to generate a container. The image contains all of the requirements for a service or application to run. Images can be very 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 operating system environment, such as a minimal Oracle Linux release.

Images can be tagged so that it is possible to identify different versions of the same image. Usually an image might include a default tag of `latest` so that Podman users can quickly and easily identify the most recent version of the image.

Images are frequently 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 Chapter 9, *Using Container Registries*.

In some cases, you may wish to modify an existing image or create your own images, which can be done by using the Buildah utility. See Chapter 7, *Building Images With Buildah* for more information.

Search 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 Chapter 9, *Using Container Registries*

Pull 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, you should specify the image reference as follows:

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

The *`registry.host`* is the resolvable hostname of the registry where the image is located. Although this is generally required, if the registry is already listed within your configuration you do not need to specify this value. The *`area`* is optional and depends on how images are stored and located 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 Appendix A, *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 registry/repository/image:tag
```

For example, to download the `oraclelinux:7-slim` image from the `os` repository on 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
```

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

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

The `podman` package provided on the Oracle Linux dnf server is configured to search Oracle Container Registry for images by default, so not specifying a domain in the same example would produce the same result:

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

The image is downloaded into the local container image store. This storage is described in more detail in Chapter 4, *Configuring Storage for Podman*.

Inspect an image

After 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": "d788eca028a0f49b6bc70b251c8535b16ee5bd94e0ab375ca8f3a923e6ce4281",
        "Digest": "sha256:94f48ee59a172eeadec3867a696982f0df4744725e2d2ec12da9c8651eec391d",
        "RepoTags": [
            "container-registry.oracle.com/os/oraclelinux:7-slim"
        ],
        "RepoDigests": [
            "container-registry.oracle.com/os/oraclelinux@sha256:94f48ee59a172eeade..."
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2020-06-10T18:22:14.311673236Z",
        "Config": {
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/bash"
            ]
        },
        "Version": "18.09.7",
        "Author": "Oracle Linux Product Team <ol-ovm-info_ww@oracle.com>",
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 244499430,
        "VirtualSize": 244499430,
        "GraphDriver": {
            "Name": "overlay",
            "Data": {
                "UpperDir": "/var/lib/containers/storage/overlay/2f915858a916.../diff",
                "WorkDir": "/var/lib/containers/storage/overlay/2f915858a9165.../work"
            }
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:2f915858a91657b11434fbb914490df4047e6f3a23e695f1357df0a1ff9019df"
            ]
        },
        "Labels": null,
        "Annotations": {},
        "ManifestType": "application/vnd.docker.distribution.manifest.v2+json",
        "User": "",
        "History": [
            {
                "created": "2018-08-30T21:49:27.028879762Z",
                "created_by": "/bin/sh -c #(nop)  MAINTAINER Oracle Linux Product Team <ol...",
                "author": "Oracle Linux Product Team <ol-ovm-info_ww@oracle.com>",
                "empty_layer": true
            },
            {
                "created": "2020-06-10T18:22:13.925457088Z",
                "created_by": "/bin/sh -c #(nop) ADD file:ac77b0540aff28a67ddde9... in / ",
                "author": "Oracle Linux Product Team <ol-ovm-info_ww@oracle.com>"
            },
            {
                "created": "2020-06-10T18:22:14.311673236Z",
                "created_by": "/bin/sh -c #(nop)  CMD [\"/bin/bash\"]",
                "author": "Oracle Linux Product Team <ol-ovm-info_ww@oracle.com>",
                "empty_layer": true
            }
        ]
    }
]
```

List available images

To list all of the locally stored container images that you previously downloaded:

```
podman images
container-registry.oracle.com/os/oraclelinux   7        d788eca028a0   21 hours ago   244 MB
container-registry.oracle.com/os/oraclelinux   latest   d788eca028a0   21 hours ago   244 MB
container-registry.oracle.com/os/oraclelinux   8        de317d637475   5 days ago     442 MB
```

Delete an image

To delete a locally stored container image that you previously downloaded, use the `podman rmi` command:

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

Note

You cannot delete an image if it is currently in use by a container, even if the container is not running. You must remove all of the containers that depend on the image before you can remove the image itself.

3.3 Managing Containers

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

Containers share namespaces and are able to access shared port mappings to communicate with each other and with the host system. Podman introduces the concept of pods to the runtime environment. Pods can be used to introduce further isolation to a group of containers and to make it easier to manage a set of services that work together to provision a logical application. See Section 3.4, “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 container-registry.oracle.com/os/oraclelinux:7-slim
```

If the image does not 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 should belong to a particular pod or whether it should use 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 is provisioned and destroy in a single step by using the `podman run` command with the `--rm` flag, for example:

```
podman run --rm container-registry.oracle.com/os/oraclelinux:7-slim cat /etc/oracle-release
Oracle Linux Server release 7.8
```

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 container-registry.oracle.com/os/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
[root@dcbe94cd0301 /]#
```

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 \
container-registry.oracle.com/os/oraclelinux:7-slim \
su -c 'yum install httpd && httpd -Dforeground'
3a430d30612d0b2c0320b207fe9ebf4d63a7e07172bb251df355d36574f3f661
```

If you run a container that does not already exist, it is created automatically. If the image that the container should use is not available locally, Podman searches the remote registries for a matching image and pulls the image automatically.

List and monitor containers

You can list all of the currently 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         S
3a430d30612d  container-registry.oracle.com/os/oraclelinux:7-slim  su -c yum install...  20 seconds ago  U
dcbe94cd0301  container-registry.oracle.com/os/oraclelinux:7-slim  /bin/bash             58 seconds ago  E
726d97e9bfbd  container-registry.oracle.com/os/oraclelinux:7-slim  top                   3 days ago      C
```

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

```
podman logs oracledaemon
```

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
c4e296d0c78ee08e0876a6670671061739499d226c8ea126a628f44d1869d499
```

Running the previous command freezes all of the running processes inside a container, in their current state:

When you are 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 oracle
```

Stopping and removing containers

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

```
podman stop oracledaemon
c4e296d0c78ee08e0876a6670671061739499d226c8ea126a628f44d1869d499
```

If you need to temporarily take the server down for maintenance, you can stop every running container that has not 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
```

3.4 Managing Pods

Podman introduces the concept of the *pod* within the context of a container runtime. This concept is borrowed from Kubernetes and is not 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 like 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 is easier to set up and tear down entire application environments using atomic operations. This also helps to facilitate the creation of service wrappers to automatically start a set of containers for an application at boot. See Chapter 6, *Managing Podman Services* for more information.

3.4.1 Creating and Managing Pods

Create a new 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 refer to a particular hostname when connecting to each other.

Pods can also be created automatically when a container is run for the first time. See Section 3.4.2, “Using Containers Within a Pod” for more information.

List all of the currently 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 of the containers within the pod have been removed, with the exception of the infrastructure container. By default, an infrastructure container is usually 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 using the `podman pod inspect` command.

3.4.2 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 are able to use a reserved port within the pod without any port mapping required. Furthermore, the containers can both use the `localhost` network name space and are able to access each other as if they were running on the same host. This example provides a simple 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 does not 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 of the containers on a host with the pod they are attached to 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
```

Chapter 4 Configuring Storage for Podman

Table of Contents

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 Inititiative (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.

It is important to understand that 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.

It is also possible to alter the configuration for Podman storage to facilitate additional requirements for a particular use case.

4.1 Setting Storage Configuration Options

For the root user, storage configuration information is located in `/etc/containers/storage.conf`.

For standard users, the storage configuration file is located at `$HOME/.config/containers/storage.conf`.

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)` man page. The following descriptions may 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 is capable of layering, Oracle only supports the `overlay2` driver in conjunction with the tested file systems.

- **graphroot.** The storage location where images are stored. As already mentioned, for standard users these are typically located in `$HOME/.local/share/containers/storage/`. A legitimate use case to change this, is where home directories may 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 additional storage locations for alternate 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 writeable content for a container. This data is temporary and exists for the lifetime of the container. For a root user, this is usually stored in `/var/run/containers/storage`.

Depending on the storage driver that you use, different storage options may 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 are able to set a `size` parameter to apply quotas to your container images.

4.2 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 where system-wide private keys, certificates, or authentication credentials may be needed during a build process to facilitate access to external resources that a user may not usually have at their privelege level.

You can define your own 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)` man 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
```

There are different options available for a volume mount. Notably the `-z` option helps where you might need to share an SELinux security context between multiple containers or between the container and the host system. This is particularly useful when running a container as a non-root user. See the `PODMAN-RUN(1)` manual page for more information on `--volume` options.

Chapter 5 Configuring Networking for Podman

Table of Contents

Use the following information 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 they are run by the root user or whether they are run by a standard user on the host system. This is because while the root user has considerably more power to modify network infrastructure on the host, the standard user has very little ability to alter network infrastructure.

In all cases containers running within a pod or a group share the same networking name space and therefore have access to the same IP addressing, MAC addressing and port mappings as each other. This shared name space makes it relatively easy to facilitate network communication between different containers or between the host and the containers running on it. You can read more about pods in Section 3.4.1, “Creating and Managing Pods”.

5.1 Configuring Proxy Server Settings

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

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

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

5.2 Configuring Networking for Standard User Containers

Since a standard user may not have the permissions to modify network infrastructure on a host, Podman does not allocate IP addressing to containers that are run by the standard user. Instead, Podman relies on port mapping to use the existing network infrastructure available on the host system. In this sense, you do not need to configure specific network settings for containers run in this way because Podman handles this automatically by performing port forwarding to container-based services.

Port publishing for a non-root user is limited to port numbers above the privileged port numbers, greater than port 1024. Therefore, when creating containers using a specified port mapping, ensure that the port number is set to a value higher than 1024. For example, to map port *`8080`* on the host to the container port 80, you could run:

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

You can also use the `-P` option when running the container to allow Podman to automatically configure port mappings, but these may be less predictable than you intend.

Once a port mapping is established, you can access the port directly from the host where the container is running. For example, in this case, the host is able to 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 ports -a` command to view all port mappings for all of the containers running on the host.

Because the containers and the host share the same network name space, a container is able to 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. To extend the example, you could run a second container by using the following command, where *`198.51.100.10`* is the IP address of the host system:

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

It is equally possible for external hosts to connect to the container port mapping by using the host's IP address and the mapped port. However, be aware that if the host has firewall software running, you may need to open the firewall port for it to be externally accessible.

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.

5.3 Configuring Networking for Root User Containers

Podman continues to facilitate port forwarding for containers that are run by the root user in exactly the same manner as containers that are run by standard users. You are still able to connect with a running container on the network you have created from the local host, or between two containers inside the same pod, by using the assigned local port mapping for that container, as described in Section 5.2, “Configuring Networking for Standard User Containers”. This is important for situations where you may need to connect to a service in a container directly from the host, or from an external system.

However, where IP assignment may be required, and where the container may need to take advantage of particular features within the network stack to communicate with other containers in a pod, containers can achieve the more comprehensive networking that is available to Podman when run by the root user.

When containers are run by the root user, Podman makes use of the Container Network Interface (CNI) tools and configuration to implement a bridged network stack that can facilitate IP address assignment and full network access for each container. CNI also includes a plug-in to facilitate DNS configuration. CNI uses IP masquerading to keep pod networks independent of each other and to facilitate IP address assignment from a set IP range. See https://github.com/containernetworking/cni for more information.

5.3.1 Using the Container Network Interface

With the exception of the default network name, all network configuration is handled within the `/etc/cni/net.d/` directory. The default network configuration is stored in `/etc/cni/net.d/87-podman-bridge.conflist`. The default network name is defined in the `/usr/share/containers/libpod.conf` file:

```
cni_default_network = "podman"
```

A new configuration file is generated in the `/etc/cni/net.d/` directory for each network that you create within Podman. In most instances, you do not need to edit or manage the files within these directories, as Podman is capable of handling the files on your behalf.

Warning

The `podman network` commands that are described here only work for containers with root permissions. If you try to use these commands without root permissions or with standard user containers, the command returns an error code.

Create a new network

Use the `podman network create` command to generate a new network configuration:

```
sudo podman network create
/etc/cni/net.d/cni-podman1.conflist
```

Podman automatically defines network settings based on the default network and any other existing networks. However, options are available to set the network range and subnet size. Use the `podman help network create` command to obtain more information about these options.

List networks

List all the Podman networks you have created:

```
sudo podman network ls
NAME          VERSION   PLUGINS
cni-podman1   0.4.0     bridge,portmap,firewall
```

Remove a network

Remove a Podman network:

```
sudo podman network rm network-name
```

Connecting containers

The network is started and stopped automatically when you create and delete containers and containers are automatically assigned IP addresses within the range that is defined for a network. You can determine the IP address that is assigned to a container by using the `podman inspect` command. Fr example you can run the following command:

```
sudo sudo podman inspect container_id | grep -i IPAddress
```

Chapter 6 Managing Podman Services

Table of Contents

Podman facilitates integration with Systemd services to make it easier 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 to other services that may run on the host system.

Podman provides the tools to automatically generate Systemd service wrapper configuration files for each of your containers or pods so that you can manage your container infrastructure using Systemd.

You can use Systemd user services if you are running your containers as a standard user, or you can configure system level services if you are running containers as the root user.

6.1 Setting SELinux Permissions for Container and Pod Service Wrappers

If you have set SELinux to `enforcing` mode on your 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
```

6.2 Generating Podman Service Wrappers

Instead of writing your own Systemd service wrapper, use the `podman generate systemd` command to automatically generate the service configuration file.

If you intend to run containers as the root user, you should store the container service wrapper configuration files in `/etc/systemd/user/`. 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

In the case where you wish to generate all of the service wrapper configuration files for the containers within a pod and for the pod itself, you can use the `podman generate systemd` with the `--files` option to automatically generate each of the service wrapper files. If you use this option, you should change to the directory where you intend to generate the files first. For example:

```
cd $HOME/.config/systemd/user/
```

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

In this case, service wrapper configuration files are generated for each container within the pod, as well as for the pod itself. The service wrapper that is responsible for the pod, includes dependencies on each of the container wrappers that are required for the pod to run successfully.

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

6.3 Managing Podman Services

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

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

If you are running your 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 this is the case, 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/user/`:

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

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

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

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

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

You can restart the service wrapper for your container or pod by using the `systemctl restart` command:

```
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:

```
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 your containers or pods with the `systemctl status` command. For example:

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

6.4 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:

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

Set the service wrapper to run automatically when the system starts up with the `systemctl enable` command:

```
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 your services are running as a standard user, you may need to give the user permission to run processes when they are not logged in, before a service can be enabled. This is usually achieved by running the following command as root:

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

For more information, see https://docs.oracle.com/en/operating-systems/oracle-linux/8/obe-systemd-linger/

6.5 Modifying Podman Service Wrapper Configuration

The Systemd service wrapper configuration files that are generated by Podman follow standard Systemd configuration format and specification. You can modify 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 by following the instructions at https://docs.oracle.com/en/operating-systems/oracle-linux/8/osmanage/ol-sysproc.html#ol-systemd.

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.

Chapter 7 Building Images With Buildah

Table of Contents

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. Since Buildah's purpose is to build images, the `run` command behaves equivalent to a `RUN` statement within a `Dockerfile`.

This difference makes it easy to separate image builds from your production level container infrastructure, running in Podman, and to easily process existing `Dockerfile` build instructions.

Use Buildah to pull images from existing registries and to modify them to create new images with more specialized functionality. You can use Buildah with an existing `Dockerfile` that works with Oracle Container Runtime for Docker to create an image, or alternately you can pull an image directly and modify it within a container running within the Buildah environment.

It is important to understand that Podman and Buildah use the same local image store. This means that you can start containers in Podman from images that you have just 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 `Dockerfile` build instructions see *Oracle® Linux: Oracle Container Runtime for Docker User's Guide*.

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

7.1 Creating Images From Dockerfiles With Buildah

Buildah, which is designed to work directly with an existing `Dockerfile`, processes the file to build an image using the 'build using dockerfile' or `bud` command. You can use any `Dockerfile` 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 called *`imagename`* from a `Dockerfile`, navigate to the directory where the file is located and run:

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

Use the `-f` *`filename`* option if the file does not use the default `Dockerfile` filename.

After Buildah has finished building the image it is tagged and added to the local image list. You can view the list of local images to confirm that the new image is available by running:

```
sudo buildah images
```

Note that since locally hosted images are shared between Buildah and Podman, the image is immediately available for use within Podman. However you may consider pushing them to a registry where they are available to other systems where Podman is installed.

If you need to verify the image by running it within a working container, you can do so by running:

```
sudo buildah from imagename
```

The `buildah from` command creates a working container for any image. This is described in more detail in Section 7.2, "Modifying Images With Buildah".

7.2 Modifying Images With Buildah

Modify images by running them as working container instances within Buildah. You can perform changes inside the working container by issuing `buildah run` commands, equivalent to `RUN` statements within a `Dockerfile`. After you have finished making changes within the working container, generate a new image based on the current status of the working container by running the `buildah commit` command.

Setting up images and working containers

Buildah can pull images from a registry in the same way that you would pull an image using Podman. For example, you can run the following command:

```
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, you can use the `buildah from` command. For example, run the following command:

```
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
```

It does not matter whether you start a container from a local image that you have pulled using Podman or Buildah, or one that you have generated using a `Dockerfile`, or one that you started by specifying a remote image on a remote registry. In all cases, a working container is created and started.

You can check which containers are available within Buildah by running 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. Where for Podman, this command emulates the equivalent Docker command and runs a process in a new container; for Buildah, the command behaves more like a `RUN` statement from within a `Dockerfile` 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 modify it to a point where you are ready to store it as an image.

Use the `buildah run` command to run shell commands in the working container to install packages and make system changes. For example:

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

```
Oracle Linux Server release 8.2
```

You can also copy files to the working container using the `buildah copy` command:

```
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. You can make changes to the file system that reflect directly within the container. For example:

```
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 may need to match these to countainer names based on the information returned by the `buildah containers` command.

If is good practice to unmount container file systems when you are finished working on them. Use the `buildah unmount` command to do this. Note that this command is not aware of container names, so you must use the command in conjunction 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 are committed to an image. Buildah allows you to commit an image locally for immediate use by Podman, but it also provides the facility to commit to a registry where you have write access. There are a range of options that can help handle registry related access, such as authentication and certificate validation requirements. You can find out more about the options available to you by referring to the `BUILDAH-COMMIT(1)` man page.

To permanently store the changes to a working container as an image, so that you can start similar containers within the Podman environment, use the `buildah commit` command. 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
```

You can also review the build image metadata with the `buildah inspect` command, which can help to identify historical changes and other information about the image or working container. For example, you can inspect the metadata for the new image that you have created using the following command:

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

As stated previously, you can use the `buildah commit` command to commit an image to any registry for which you have write access. For example, to commit to a local registry, you would run the following command:

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

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

Removing a working container

When you have finished modifying a working container and you have generated an image from it, you can stop and remove the container from Buildah's container list using the `buildah rm` command if you did not already remove the working container when you committed the image, for example:

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

This command removes the working container and unmounts any mounts that may have been set up for it. Once the container is removed, any changes that you made to the working container are lost and it is not possible to retrieve them unless you created an image of the working container before you removed it.

It is important to note that this command only removes the working container from the Buildah environment. It does not affect any containers running under Podman and it does not affect any existing images. You should also be aware that 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.

7.3 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. 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, *`imagename`*, 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
```

Since Podman attempts to maintain syntax compatibility with the Docker CLI, you can run this command using the following syntax assuming that the local image is stored with its registry name and tag, for example:

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

Since registries are HTTP based resources, they are often protected using TLS/SSL certificates and may require authentication. There are several options related to how the command handles these requirements. Refer to the `BUILDAH-PUSH(1)` man page for more information about these options.

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 do not have access to a registry. For example, you can create a Docker compatible archive, similar to what you get if you run the `docker save` command in that environment:

```
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.

Chapter 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 does not require a running daemon to function.

Note

You do not need root permissions to run Skopeo commands. If you encounter errors, ensure that you have configured the appropriate proxy server settings, and that you have the necessary access permissions for the remote registries you are using.

For more information, see the `SKOPEO(1)` man page.

Inspect an image in a remote registry with Skopeo

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 environment variables set for the image. This can also be useful for inspecting alternate available tags for a matching image name within the registry. For example:

```
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"
    ]
}
```

You can review the default configuration settings and build history for an image by adding the `--config` flag:

```
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-INSPECT(1)` man page.

Copy an image between container storage types with Skopeo

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 your local Podman container storage by using the `containers-storage:` prefix:

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

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

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

In that 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)` man page.

Delete an image from container storage with Skopeo

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

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

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

Chapter 9 Using Container Registries

Table of Contents

This chapter describes how to sign in to Oracle Container Registry, create your own 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. A registry is used to store container images, which are used to deploy containers as required.

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

1. **container-registry.oracle.com.** Oracle Container Registry, which contains licensed and open source Oracle software. The Oracle Container Registry is located at https://container-registry.oracle.com.

   The Oracle Container Registry provides a web interface to browse and select the images for the software that your organization can use.

   To use licensed Oracle software images, first log into the Oracle Container Registry web interface and accept the *Oracle Standard Terms and Restrictions* for the required software images.

   Open source software images and all of the software the image contains is licensed under one or more open source license provided in the container image. Use of the container image is subject to the terms of those licenses.

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

2. **quay.io.** The Quay Container Registry is a broadly used registry provided by Red Hat. A large number of open source images are avaialable at this registry.

3. **docker.io.** The Docker Hub registry provides a very large number of 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 available on the Docker Hub.

Enterprise environments may additionally consider setting up a local container registry. This provides the opportunity to convert customized containers into images that can be committed into a local registry, to be used for future container deployment, reducing the amount of customized configuration that may need to be performed for mass deployments. A local registry can also cache and host images pulled from an upstream registry. This can reduce network overhead and latency when deploying matching containers across a spread of local systems.

9.1 Registry Configuration

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

The configuration file is well commented to explain the options that are available. The registries that are searched when you attempt to pull or use an image, that is not 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 are defined in this list. When a local registry is used, it is common to add it as the first entry in this list, so that it is searched and used first.

To use an insecure registry without a valid SSL certificate or that does not use SSL at all, add the registry domain name to the `registries` list within the `[registries.insecure]` configuration block.

9.1.1 Configuring Podman for Signed Images

You can configure Podman to only trust images from a remote registry if they are 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 on your infrastructure.

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 is accessible via HTTPS. A public GPG key 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 your 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 at `/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 located at `/etc/containers/policy.json`. Registry configuration appears under the `docker` key, which you may 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 your configuration is correct by pulling an image from the remote registry. See Section 9.2, "Pulling Images From the Oracle Container Registry" for an example. Note that if the signature requirement is configured correctly, there is no difference in the 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 an alternate 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}
```

9.2 Pulling Images From the Oracle Container Registry

If you are pulling a licensed Oracle software image, you must first log into 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 Section 9.2.1, "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 lower case. Note that when referencing images, Oracle recommends that you always specify the appropriate *`tag`* to use.

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 it should be run may also be available on the same page.

9.2.1 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 do not need to log into the Oracle Container Registry or accept the *Oracle Standard Terms and Restrictions* to pull open source Oracle software images.

To pull a licensed software image from the Oracle Container Registry:

1. In a web browser, log into the Oracle Container Registry using your Oracle Account at https://container-registry.oracle.com.

2. Use the web interface to accept the *Oracle Standard Terms and Restrictions* for the Oracle software images you want to pull. Your acceptance of these terms are stored in a database that links the software images to your Oracle Account. Your acceptance of the *Oracle Standard Terms and Restrictions* is valid only for the repositories for which you accept the terms. You may need to repeat this process if you attempt to pull software from alternate or newer repositories in the registry. This is subject to change without notice.

3. Use the web interface to browse or search for Oracle software images.

4. On the host system, use the `podman login` command to authenticate against the Oracle Container Registry by using the same Oracle Account that you used to log into the web interface:

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

   You are prompted for the user name and password for the Oracle Account.

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 Section 9.2, “Pulling Images From the Oracle Container Registry”.

   If your Oracle Account credentials can be verified and the *Oracle Standard Terms and Restrictions* have been accepted, 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, it is good practice to log out of the registry to prevent unauthorized access, and to remove any record of your credentials that Podman may store for future operations:

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

9.2.2 Using the Oracle Container Registry Mirrors With Podman

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

To get a list of the available mirrors, and the command to pull the image from the mirror, see the information page for an image using the Oracle Container Registry web interface. The list of registry mirrors is available towards the end of the image information page, in the `Tags` table. The table heading includes a `Download Mirror` drop down 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.

Pull an image from an Oracle Container Registry mirror using the URL for that mirror. For example, to pull the Oracle Linux 7 image from the Sydney mirror, use:

```
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 located at https://container-registry.oracle.com .

To pull licensed Oracle software images, log 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
```

When a mirror is used regularly, add it to the configuration so that it is used by default for searches and pull requests. See Section 9.1, “Registry Configuration” for more information.

9.3 Using the Docker Hub With Podman

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

You are able to browse the Docker Hub and to pull some images from the hub anonymously, but to access the majority of images hosted there, you must log in with a valid Docker ID. If you do not have a Docker ID, you can register at https://hub.docker.com/signup.

The Docker Hub provides a web interface that allows you to select the Docker Certified images that you want to install. For some images, you may need to click on `Proceed to Checkout` button agree to any terms and conditions that may apply, or to make payment if required before you are able to access the image.

When you have agreed to the terms and conditions that apply to an image, the image is stored in the `My Content` area, so that you can revisit it later.

Each image provides a description and set up instructions.

The Docker Hub may require that you are logged in before you can inspect or pull particular images hosted in this registry. This makes sure the terms and conditions that apply to the image have been accepted, and that any payments have been settled. The following example illustrates how you can log into the Docker Hub, and 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
```

9.4 Setting up a Local Container Registry

This section contains information about setting up a local container registry server, which can be used to host your own 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. This is usually located at `/var/lib/registry` although you may select an alternate path if you intend to run the registry as a standard user. It is good practice to create a separate file system for this. It is recommended you create a Btrfs formatted file system to allow you to easily scale your registry file system, and to leverage Btrfs features such as snapshotting.

9.4.1 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 does not 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*.

If you want to disable X.509 certificate validation for testing purposes, see Section 9.1, “Registry Configuration”.

To use the X.509 Certificate with Podman:

1. If the host's X.509 certificate was issued by an intermediate Certificate Authority (CA), you must combine the host's certificate with the intermediate CA's certificate to create a chained certificate so that Docker can 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, into which you need to copy 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
   ```

   where *`certfile`* is the full path to the host's X.509 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. Make sure the file permissions are correct for the private key:

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

9.4.2 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 is started.

You can map an alternate port number for your container registry, if required, by changing the *`5000`* in the command above to match the port number that you would prefer to use.

If you do not have an Oracle Account and if you do not require support, you can alternately use the publicly available container registry image at `docker.io/library/registry:latest`.

9.4.3 Setting up the Registry Port

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

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

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

If you do not run the registry on the default port you can specify the port directly:

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

9.4.4 Distributing X.509 Certificates

If the registry host uses a self-signed X.509 certificate, you must distribute the certificate to **all** hosts in your deployment that you intend to use the local container registry. Certificates for each registry are stored in `/etc/containers/certs.d/`*`registry_hostname`*`:`*`port`*`/` for the root user. 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 using:

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

9.4.5 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 may either pull images from a registry, such as the Oracle Container Registry, and then commit them to your local registry, or you may wish to create your own images based on upstream images.

To import images into a local container registry:

1. Pull an image from a registry. For example, you can pull an image from the Oracle Container Registry:

   ```
   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:5000/ol7image:v1
   ```

   In this example, `localhost` is the hostname where the local registry is located and `5000` is the port number that the registry listens on. If you are working on a Podman installation located 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 lower case to be a valid tag.

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

   ```
   sudo podman push localhost:5000/ol7image:v1
   ```

See Chapter 7, *Building Images With Buildah* for more information about how you can create your own images. When you have committed a customized image, you can tag it and push it to your local registry as indicated in the steps above.

Chapter 10 Known Issues

Table of Contents

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

10.1 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 do not 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 your 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 your `$HOME/.bashrc` file to continue using using self-signed certificates where a SAN is not set:

```
export GODEBUG=x509ignoreCN=0
```

10.2 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 there are containers in the environment. Running the command when there are no containers is meaningless.

10.3 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`), as well as an input channel for passing control signals (`stdin`). Otherwise, you cannot 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.

10.4 Authentication error occurs when attempting to pull an image by specifying an incorrect name

If you attempt to pull an image by running the `podman pull` *`image-name`* command, but you do not 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.

10.5 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 does not 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 Oracle recommends using the `-slim` variants for the smallest possible image size.

For more information, see Appendix A, *Oracle Linux Container Image Tagging Conventions*.

Appendix A 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 additional 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
```

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, you should 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.

You should not directly use update level tags within your Dockerfile or within any of your 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 is only present at a particular update level of Oracle Linux.

Using an update level tag can result in your containers running unpatched software that may expose you 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 does not 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 does not provide a `latest` tag for Oracle Linux images.

The following reasons for Oracle's decision on this help to explain why this tag is not available:

- When the `latest` tag is used, it can result in significant jumps between distribution releases rather than simple update levels. This is usually 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.

- There is no easy way to identify which `latest` image was used for a particular build, 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 is not automatic and it is possible for a more recent image to be available while the image tagged as `latest` has not been updated. This can lead to unexpected consequences.

- There is no guarantee that all tools treat the `latest` tag the same. While some tools may default to always pulling an image tagged as `latest` from an upstream registry, other tools may default to a locally stored image also tagged as `latest`, even if it has fallen out of date.

This decision may 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
```