How to Manage Containers Using Podman and Skopeo in RHEL 8

James Kiarie | November 5, 2020 | Podman, RedHat | 1 Comment

AD

One of the challenges developers faced in the past is getting applications to run reliably across multiple computing environments. Oftentimes, applications didn't run as expected or encountered errors and failed altogether. And that's where the concept of **containers**was born.

What are Container Images?

Container images are static files that ship with executable code that runs in an isolated environment. A container image comprises system libraries, dependencies & other platform settings needed by the application to run in diverse environments.

Red Hat Linux provides a set of useful container tools that you can leverage to work directly with Linux containers using requiring docker commands. These include:

- **Podman** – This is a daemon less container engine for running and managing **OCI**containers in either root or rootless mode. **Podman** is similar to **Docker** and has the same command options except that **Docker** is a daemon. You can pull, run, and manage container images using **podman** in much the same way as you would with **Docker**. **Podman** comes with lots of advanced features, fully integrates with **systems,** and offers user Namespace support which includes running containers without the need for a root user.
- **Skopeo**: This is a command-line tool used for copying container images from one registry to another. You can use **Skopeo** to copy images to and from a particular host as well as copy images to another container registry or environment. Apart from copying images, you can use it to inspect images from various registries and use signatures to create and verify images.
- **Buildah**: This is a set of command-line tools used for creating and managing container **OCI** images using **Docker** files.

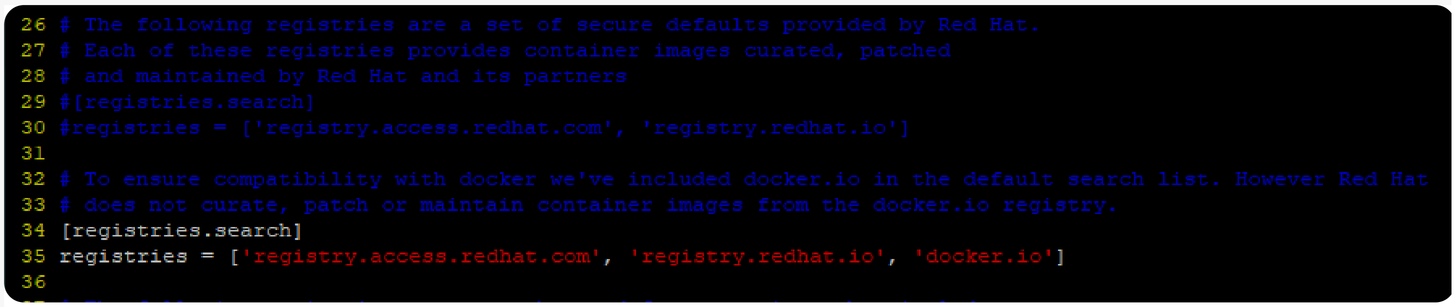
In this article, we will focus on Managing containers using **podman** and **Skopeo**.

Searching Container Images from a Remote Registry

The **podman search** command allows you to search selected remote registries for container images. The default list of registries is defined in the **registries.conf** file located in the **/etc/containers/** directory.

The registries are defined by 3 sections.

- **[registries.search]** – This section specifies the default registries that **podman** can search for container images. It searches for the requested image in the **registry.access.redhat.com**, **registry.redhat.io,** and **docker.io** registries.

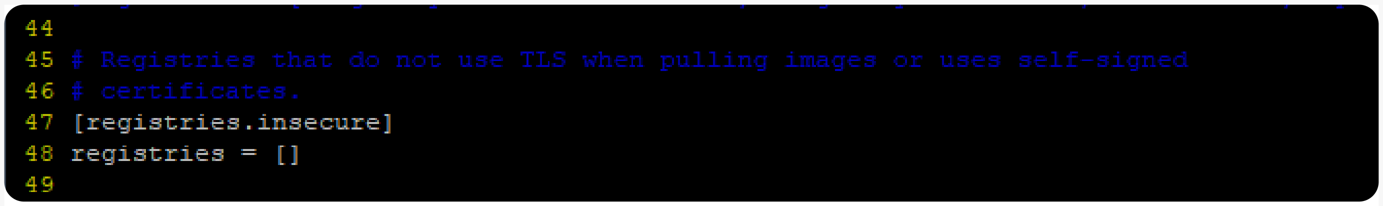


```
# The following registries are a set of secure defaults provided by Red Hat.
# Each of these registries provides container images curated, patched
# and maintained by Red Hat and its partners
#[registries.search]
#registries = ['registry.access.redhat.com', 'registry.redhat.io']

# To ensure compatibility with docker we've included docker.io in the default search list. However Red Hat
# does not curate, patch or maintain container images from the docker.io registry.
[registries.search]
registries = ['registry.access.redhat.com', 'registry.redhat.io', 'docker.io']
```

Default Registries

- **[registries.insecure]**– This section specifies registries that do not implement TLS encryption i.e insecure registries. By default, no entries are specified.

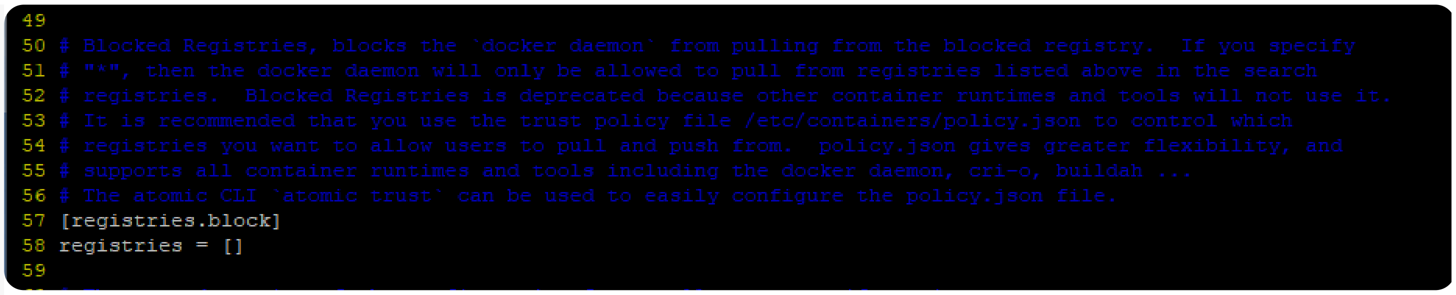


```

# Registries that do not use TLS when pulling images or uses self-signed
# certificates.
[registries.insecure]
registries = []
```

Insecure Registries

- **[registries.block]** – This blocks or denies access to the specified registries from your local system. By default, no entries are specified.



```

# Blocked Registries, blocks the `docker daemon` from pulling from the blocked registry.  If you specify
# "*", then the docker daemon will only be allowed to pull from registries listed above in the search
# registries.  Blocked Registries is deprecated because other container runtimes and tools will not use it.
# It is recommended that you use the trust policy file /etc/containers/policy.json to control which
# registries you want to allow users to pull and push from.  policy.json gives greater flexibility, and
# supports all container runtimes and tools including the docker daemon, cri-o, buildah ...
# The atomic CLI `atomic trust` can be used to easily configure the policy.json file.
[registries.block]
registries = []

```

Blocks Registries

As a regular (**non-root**) user running the podman command, you can define your own **registries.conf** file on your home directory (**$HOME/.config/containers/registries.conf**) to override system-wide settings.

Rules When Specifying Registries

As you specify the registries, keep in mind the following:

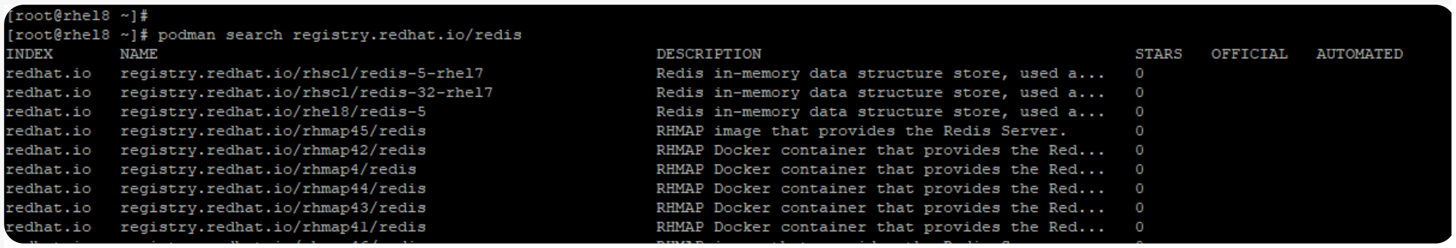
- Every registry should be enclosed by single quotes.
- Registries can be specified using either a hostname or IP address.
- If multiple registries are specified, then they should be separated by commas.
- If a registry uses a non-standard port – either port TCP ports 443 for secure and 80 for insecure, – the port number should be specified alongside the registry name e.g. **registry.example.com:5566**.

To search a registry for a container image using the syntax:

```
# podman search registry/container_image
```

For example, to search for a **Redis** image in the **registry.redhat.io** registry, invoke the command:

```
# podman search registry.redhat.io/redis
```

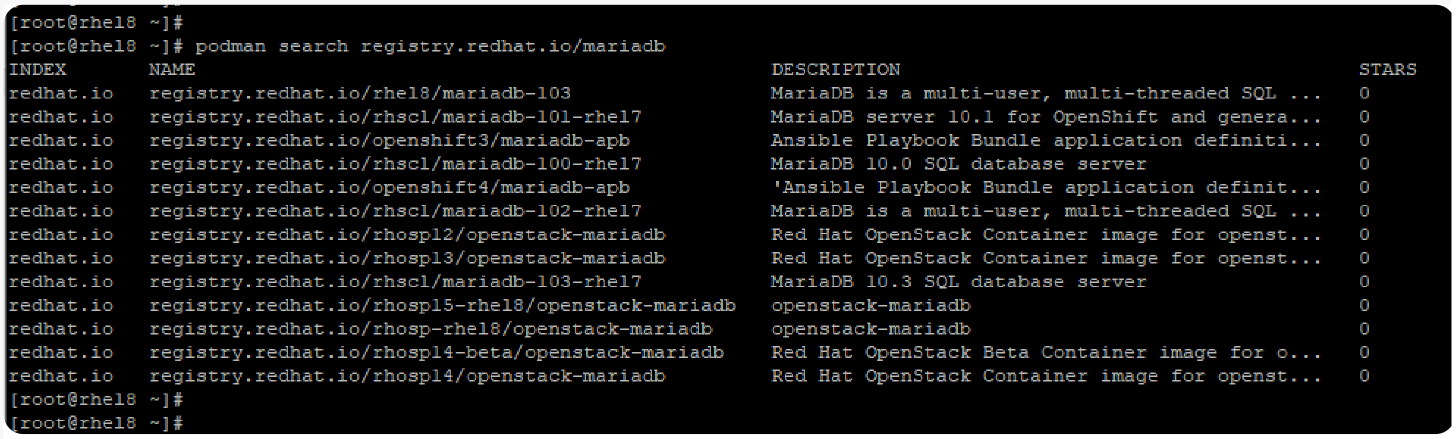


```
[root@rhel8 ~]#
[root@rhel8 ~]# podman search registry.redhat.io/redis
INDEX       NAME                                            DESCRIPTION                                       STARS   OFFICIAL   AUTOMATED
redhat.io   registry.redhat.io/rhscl/redis-5-rhel7          Redis in-memory data structure store, used a...   0
redhat.io   registry.redhat.io/rhscl/redis-32-rhel7         Redis in-memory data structure store, used a...   0
redhat.io   registry.redhat.io/rhel8/redis-5                Redis in-memory data structure store, used a...   0
redhat.io   registry.redhat.io/rhmap45/redis                RHMAP image that provides the Redis Server.       0
redhat.io   registry.redhat.io/rhmap42/redis                RHMAP Docker container that provides the Red...   0
redhat.io   registry.redhat.io/rhmap4/redis                 RHMAP Docker container that provides the Red...   0
redhat.io   registry.redhat.io/rhmap44/redis                RHMAP Docker container that provides the Red...   0
redhat.io   registry.redhat.io/rhmap43/redis                RHMAP Docker container that provides the Red...   0
redhat.io   registry.redhat.io/rhmap41/redis                RHMAP Docker container that provides the Red...   0
```

Search Registry for Container Image

To search for a **MariaDB** container image run.

```
# podman search registry.redhat.io/mariadb
```

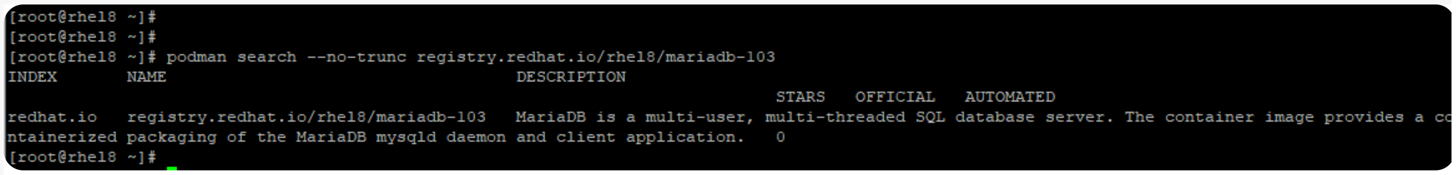


```
[root@rhel8 ~]#
[root@rhel8 ~]# podman search registry.redhat.io/mariadb
INDEX       NAME                                                DESCRIPTION                                        STARS
redhat.io   registry.redhat.io/rhel8/mariadb-103                MariaDB is a multi-user, multi-threaded SQL ...   0
redhat.io   registry.redhat.io/rhscl/mariadb-101-rhel7          MariaDB server 10.1 for OpenShift and genera...   0
redhat.io   registry.redhat.io/openshift3/mariadb-apb           Ansible Playbook Bundle application definiti...   0
redhat.io   registry.redhat.io/rhscl/mariadb-100-rhel7          MariaDB 10.0 SQL database server                   0
redhat.io   registry.redhat.io/openshift4/mariadb-apb           'Ansible Playbook Bundle application definit...   0
redhat.io   registry.redhat.io/rhscl/mariadb-102-rhel7          MariaDB is a multi-user, multi-threaded SQL ...   0
redhat.io   registry.redhat.io/rhosp12/openstack-mariadb        Red Hat OpenStack Container image for openst...   0
redhat.io   registry.redhat.io/rhosp13/openstack-mariadb        Red Hat OpenStack Container image for openst...   0
redhat.io   registry.redhat.io/rhscl/mariadb-103-rhel7          MariaDB 10.3 SQL database server                   0
redhat.io   registry.redhat.io/rhosp15-rhel8/openstack-mariadb  openstack-mariadb                                  0
redhat.io   registry.redhat.io/rhosp-rhel8/openstack-mariadb    openstack-mariadb                                  0
redhat.io   registry.redhat.io/rhosp14-beta/openstack-mariadb   Red Hat OpenStack Beta Container image for o...   0
redhat.io   registry.redhat.io/rhosp14/openstack-mariadb        Red Hat OpenStack Container image for openst...   0
[root@rhel8 ~]#
[root@rhel8 ~]#
```

Search MariaDB Container Image

To obtain an elaborate description of a container image, use the `--no-trunc` option before the name of the container image from the results that you get. For instance, we will try to obtain a detailed description of the MariaDB container image as shown:

```
# podman search --no-trunc registry.redhat.io/rhel8/mariadb-103
```



```
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# podman search --no-trunc registry.redhat.io/rhel8/mariadb-103
INDEX       NAME                                   DESCRIPTION
                                                                                     STARS   OFFICIAL   AUTOMATED
redhat.io   registry.redhat.io/rhel8/mariadb-103   MariaDB is a multi-user, multi-threaded SQL database server. The container image provides a co
ntainerized packaging of the MariaDB mysqld daemon and client application.   0
[root@rhel8 ~]#
```

List Description of MariaDB Container Image

Pulling Container Images

Pulling or retrieving container images from a remote registry requires that you first authenticate before anything else. For example, to retrieve the MariaDB container image, first log in to the Redhat registry:

```
# podman login
```

Provide your username and password and hit ‘**ENTER**‘ on your keyboard. If all goes well, you should get a confirmation message that the login to the registry was successful.

```
Login Succeeded!
```

Now, you can pull the image using the syntax shown:

```
# podman pull <registry>[:<port>]/[<namespace>/]<name>:<tag>
```

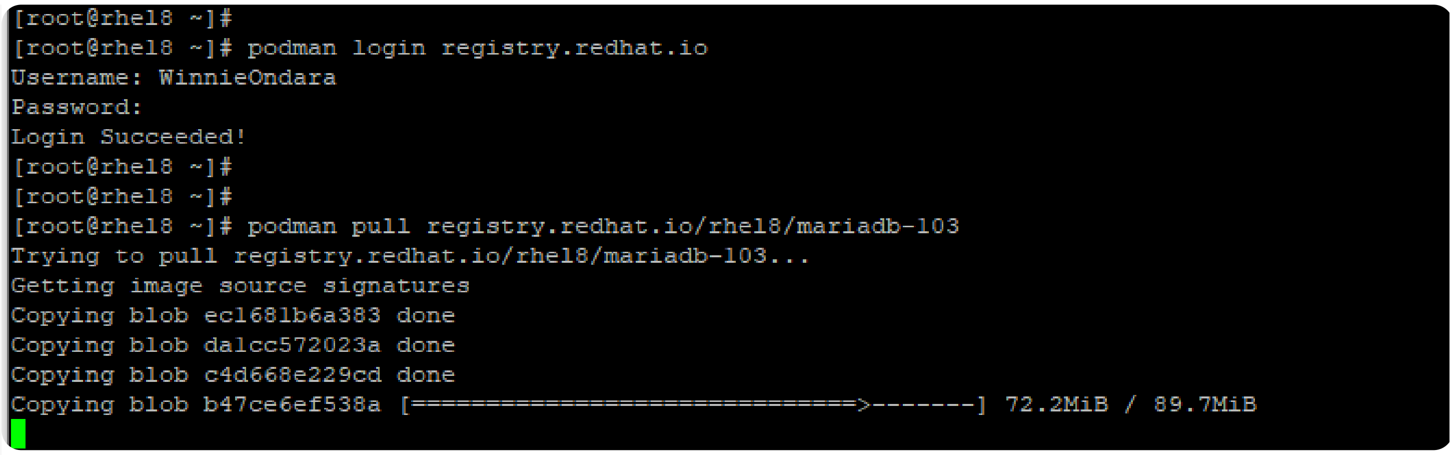
The `<registry>` refers to the remote host or registry that provides a repository of container images on the TCP `<port>` . The `<namespace>` and the `<name>` collectively specify a container image based on the `<namespace>` at the registry. Finally, the `<tag>` option specifies the version of the container image. If none is specified, the default tag – latest – is used.

It’s always recommended to add trusted registries, that is those that provide encryption and don’t allow anonymous users to spawn accounts with random names.

To pull the MariaDB image, run the command:

```
# podman pull registry.redhat.io/rhel8/mariadb-103
```

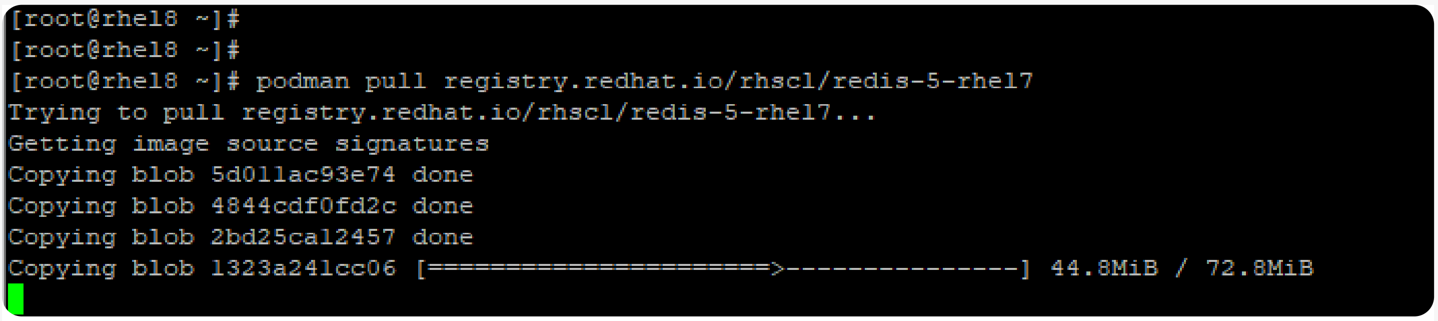
- The `<registry>` – registry.redhat.io
- The `<namespace>` – rhel8
- The `<name>` – MariaDB
- The `<tag>` – 103


```
[root@rhel8 ~]#
[root@rhel8 ~]# podman login registry.redhat.io
Username: WinnieOndara
Password:
Login Succeeded!
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# podman pull registry.redhat.io/rhel8/mariadb-103
Trying to pull registry.redhat.io/rhel8/mariadb-103...
Getting image source signatures
Copying blob ec1681b6a383 done
Copying blob da1cc572023a done
Copying blob c4d668e229cd done
Copying blob b47ce6ef538a [==============================>-------] 72.2MiB / 89.7MiB
```

Pull MariaDB Image

For subsequent container images pull, no further logging in is required since you are already authenticated. To pull a **Redis** container image, simply run:

```
# podman pull registry.redhat.io/rhscl/redis-5-rhel7
```

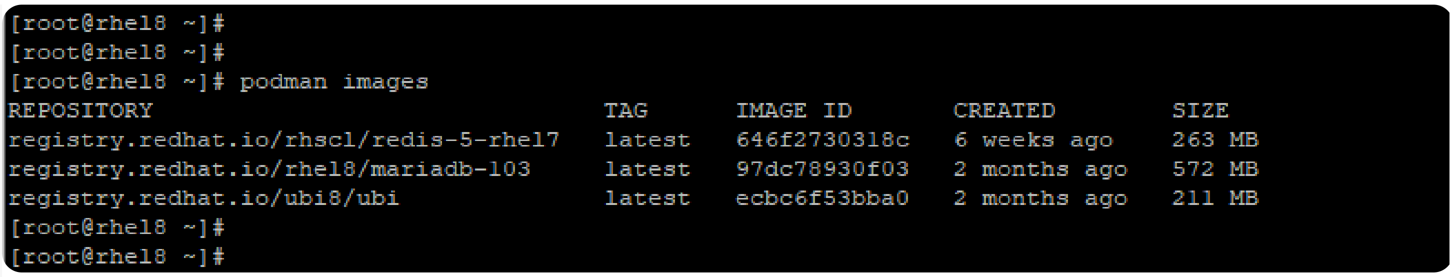

```
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# podman pull registry.redhat.io/rhscl/redis-5-rhel7
Trying to pull registry.redhat.io/rhscl/redis-5-rhel7...
Getting image source signatures
Copying blob 5d011ac93e74 done
Copying blob 4844cdf0fd2c done
Copying blob 2bd25ca12457 done
Copying blob 1323a241cc06 [======================>---------------] 44.8MiB / 72.8MiB
```

Pull Redis Container Image

Listing Container Images

Once you are done pulling the images, you can view the images currently existing on your host by running the podman images command.

```
# podman images
```



```
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# podman images
REPOSITORY                              TAG      IMAGE ID       CREATED         SIZE
registry.redhat.io/rhscl/redis-5-rhel7  latest   646f2730318c   6 weeks ago     263 MB
registry.redhat.io/rhel8/mariadb-103    latest   97dc78930f03   2 months ago    572 MB
registry.redhat.io/ubi8/ubi             latest   ecbc6f53bba0   2 months ago    211 MB
[root@rhel8 ~]#
[root@rhel8 ~]#
```

List Container Images

Inspecting Container Images

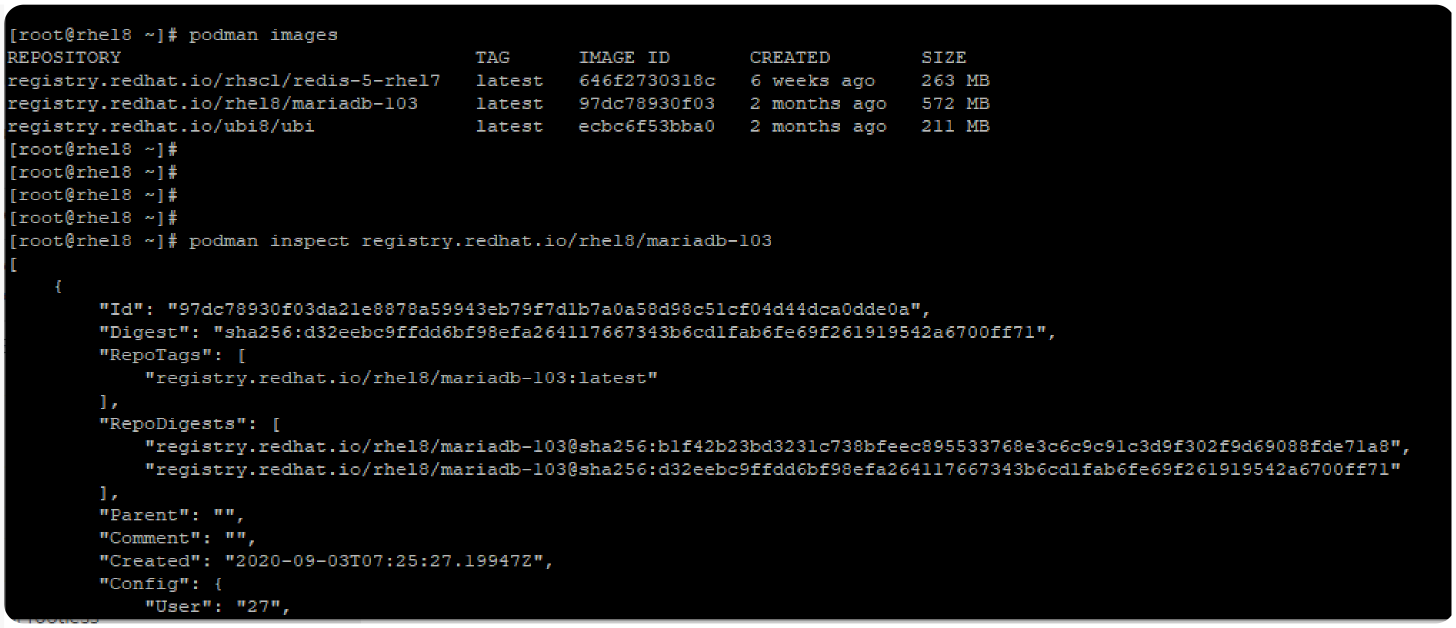
Before running a container, it's always a good idea to probe the image and get to understand what it does. The **podman inspect** command prints out a sea of metadata about the container such as the OS and Architecture.

To inspect an image, run the **podman inspect** command followed by the image ID or repository.

```
# podman inspect IMAGE ID
OR
# podman inspect REPOSITORY
```

In the example below, we're inspecting the **MariaDB** container.

```
# podman inspect registry.redhat.io/rhel8/mariadb-103
```



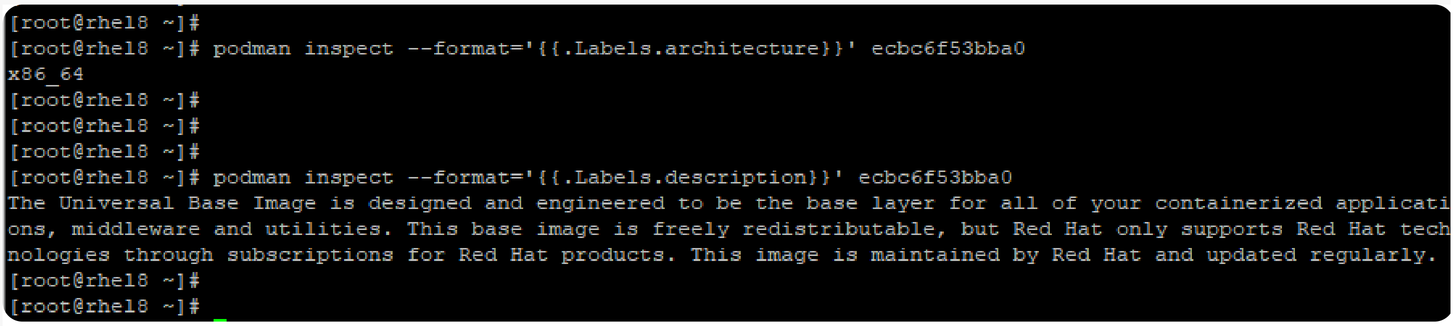
```
[root@rhel8 ~]# podman images
REPOSITORY                              TAG      IMAGE ID       CREATED        SIZE
registry.redhat.io/rhscl/redis-5-rhel7  latest   646f2730318c   6 weeks ago    263 MB
registry.redhat.io/rhel8/mariadb-103    latest   97dc78930f03   2 months ago   572 MB
registry.redhat.io/ubi8/ubi             latest   ecbc6f53bba0   2 months ago   211 MB
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# podman inspect registry.redhat.io/rhel8/mariadb-103
[
    {
        "Id": "97dc78930f03da21e8878a59943eb79f7d1b7a0a58d98c51cf04d44dca0dde0a",
        "Digest": "sha256:d32eebc9ffdd6bf98efa264117667343b6cd1fab6fe69f261919542a6700ff71",
        "RepoTags": [
            "registry.redhat.io/rhel8/mariadb-103:latest"
        ],
        "RepoDigests": [
            "registry.redhat.io/rhel8/mariadb-103@sha256:b1f42b23bd3231c738bfeec895533768e3c6c9c91c3d9f302f9d69088fde71a8",
            "registry.redhat.io/rhel8/mariadb-103@sha256:d32eebc9ffdd6bf98efa264117667343b6cd1fab6fe69f261919542a6700ff71"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2020-09-03T07:25:27.19947Z",
        "Config": {
            "User": "27",
```

Inspecting MariaDB Container Images

To pull specific metadata for a container pass the `--format` option followed by the metadata and the container identity (Image ID or name).

In the example below, we're retrieving information about the architecture and description of the RHEL 8 base container which falls under the '**Labels**' section.

```
# podman inspect --format='{{.Labels.architecture}}' image ID
# podman inspect --format='{{.Labels.description}}' image ID
```

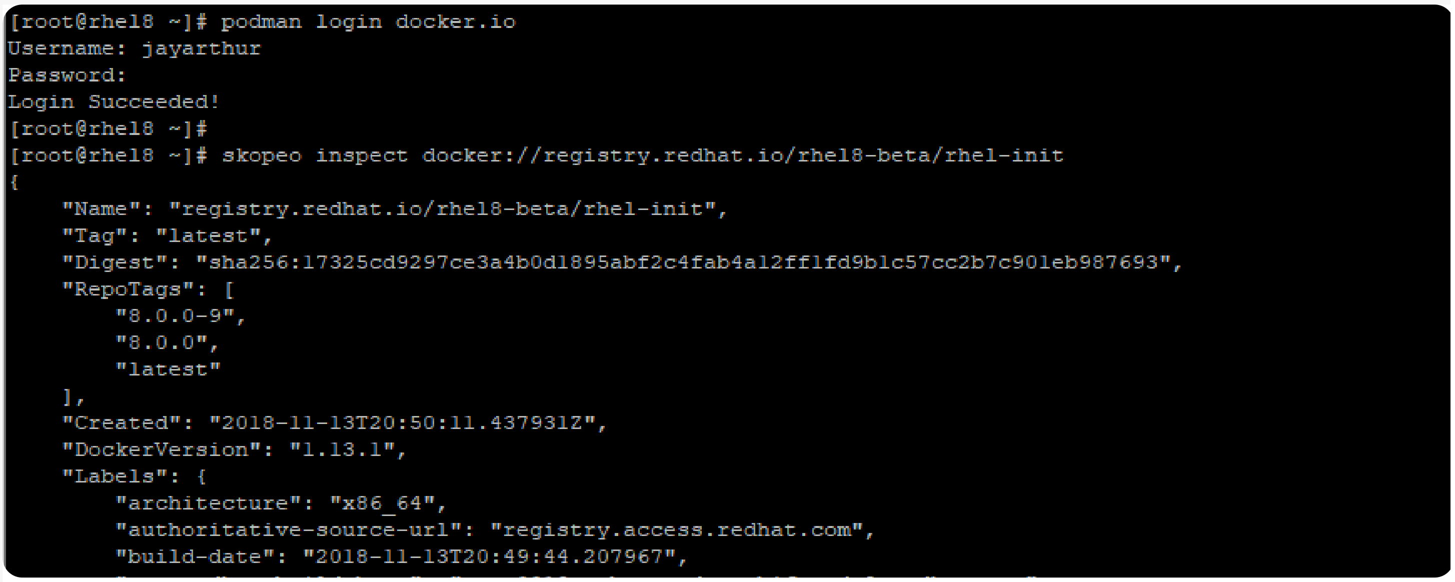


```
[root@rhel8 ~]#
[root@rhel8 ~]# podman inspect --format='{{.Labels.architecture}}' ecbc6f53bba0
x86_64
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# podman inspect --format='{{.Labels.description}}' ecbc6f53bba0
The Universal Base Image is designed and engineered to be the base layer for all of your containerized applicati
ons, middleware and utilities. This base image is freely redistributable, but Red Hat only supports Red Hat tech
nologies through subscriptions for Red Hat products. This image is maintained by Red Hat and updated regularly.
[root@rhel8 ~]#
[root@rhel8 ~]#
```

Get Info About Container Architecture

To inspect a remote image from another registry, use the **skopeo inspect** command. In the example below, we are inspecting an RHEL 8 init image hosted on **Docker**.

```
# skopeo inspect docker://registry.redhat.io/rhel8-beta/rhel-init
```



```
[root@rhel8 ~]# podman login docker.io
Username: jayarthur
Password:
Login Succeeded!
[root@rhel8 ~]#
[root@rhel8 ~]# skopeo inspect docker://registry.redhat.io/rhel8-beta/rhel-init
{
    "Name": "registry.redhat.io/rhel8-beta/rhel-init",
    "Tag": "latest",
    "Digest": "sha256:17325cd9297ce3a4b0d1895abf2c4fab4a12ff1fd9b1c57cc2b7c901eb987693",
    "RepoTags": [
        "8.0.0-9",
        "8.0.0",
        "latest"
    ],
    "Created": "2018-11-13T20:50:11.437931Z",
    "DockerVersion": "1.13.1",
    "Labels": {
        "architecture": "x86_64",
        "authoritative-source-url": "registry.access.redhat.com",
        "build-date": "2018-11-13T20:49:44.207967",
```

Inspect Remote Image from Docker Registry

Tagging Container Images

As you might have noted, image names are usually generic in nature. For example, the redis image is labeled:

```
registry.redhat.io/rhscl/redis-5-rhel7
```

Tagging images gives them a more intuitive name to better understand what they contain. Using the **podman tag** command, you can create an image tag which is essentially an alias to an image name that comprises different parts.

These are:

```
registry/username/NAME:tag
```

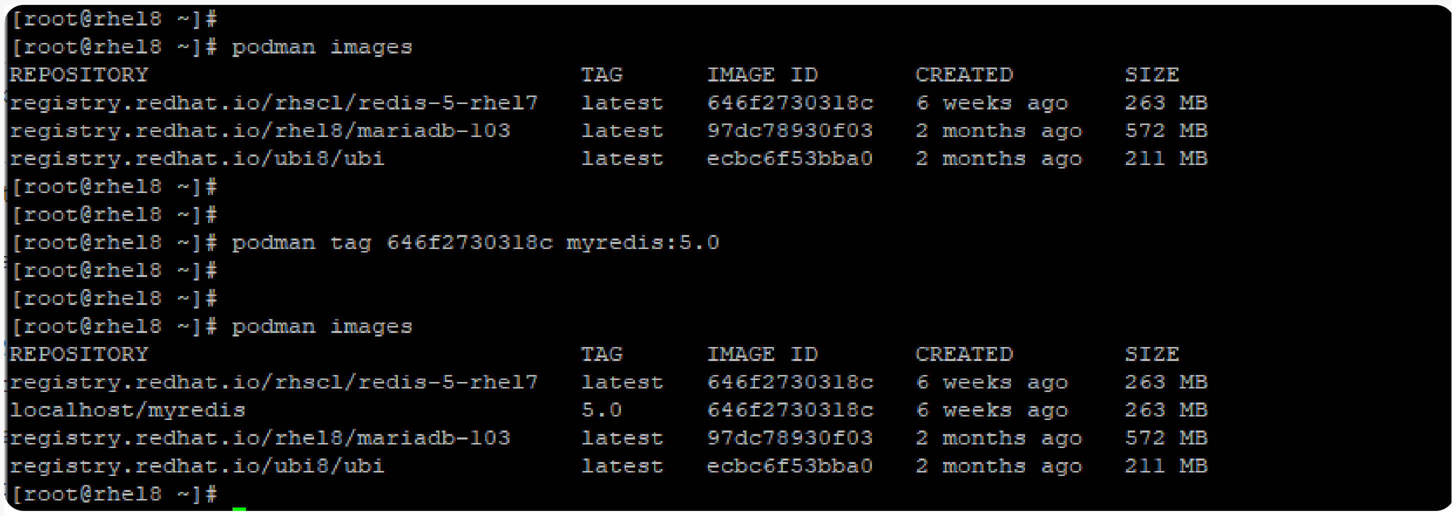
For example, to change the generic name of the Redis image which has an ID of **646f2730318c** , we will execute the command:

```
# podman tag 646f2730318c myredis
```

To add a tag at the end append a full colon followed by the tag number:

```
# podman tag 646f2730318c myredis:5.0
```

Without adding the tag number, it will just be assigned the attribute latest.



```
[root@rhel8 ~]#
[root@rhel8 ~]# podman images
REPOSITORY                               TAG      IMAGE ID       CREATED         SIZE
registry.redhat.io/rhscl/redis-5-rhel7   latest   646f2730318c   6 weeks ago     263 MB
registry.redhat.io/rhel8/mariadb-103     latest   97dc78930f03   2 months ago    572 MB
registry.redhat.io/ubi8/ubi              latest   ecbc6f53bba0   2 months ago    211 MB
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# podman tag 646f2730318c myredis:5.0
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# podman images
REPOSITORY                               TAG      IMAGE ID       CREATED         SIZE
registry.redhat.io/rhscl/redis-5-rhel7   latest   646f2730318c   6 weeks ago     263 MB
localhost/myredis                        5.0      646f2730318c   6 weeks ago     263 MB
registry.redhat.io/rhel8/mariadb-103     latest   97dc78930f03   2 months ago    572 MB
registry.redhat.io/ubi8/ubi              latest   ecbc6f53bba0   2 months ago    211 MB
[root@rhel8 ~]#
```

Set Name for Redis Container Image

Running Container Images

To run a container, use the **podman run** command. For example:

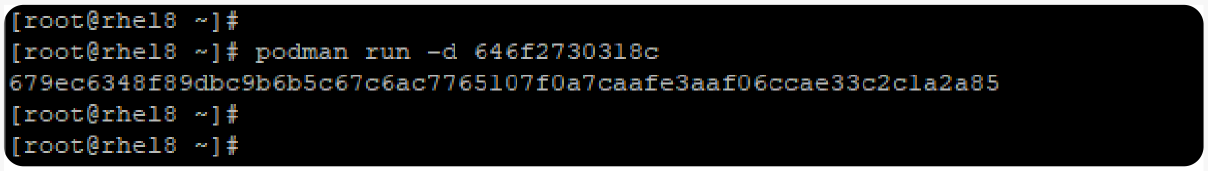
```
# podman run image_id
```

To run a container silently in the background as a daemon service use the `-d` option as shown.

```
# podman run -d image_id
```

For example, to run the **redis** image with ID **646f2730318c**, we will invoke the command:

```
# podman run -d 646f2730318c
```



```
[root@rhel8 ~]#
[root@rhel8 ~]# podman run -d 646f2730318c
679ec6348f89dbc9b6b5c67c6ac7765107f0a7caafe3aaf06ccae33c2c1a2a85
[root@rhel8 ~]#
[root@rhel8 ~]#
```

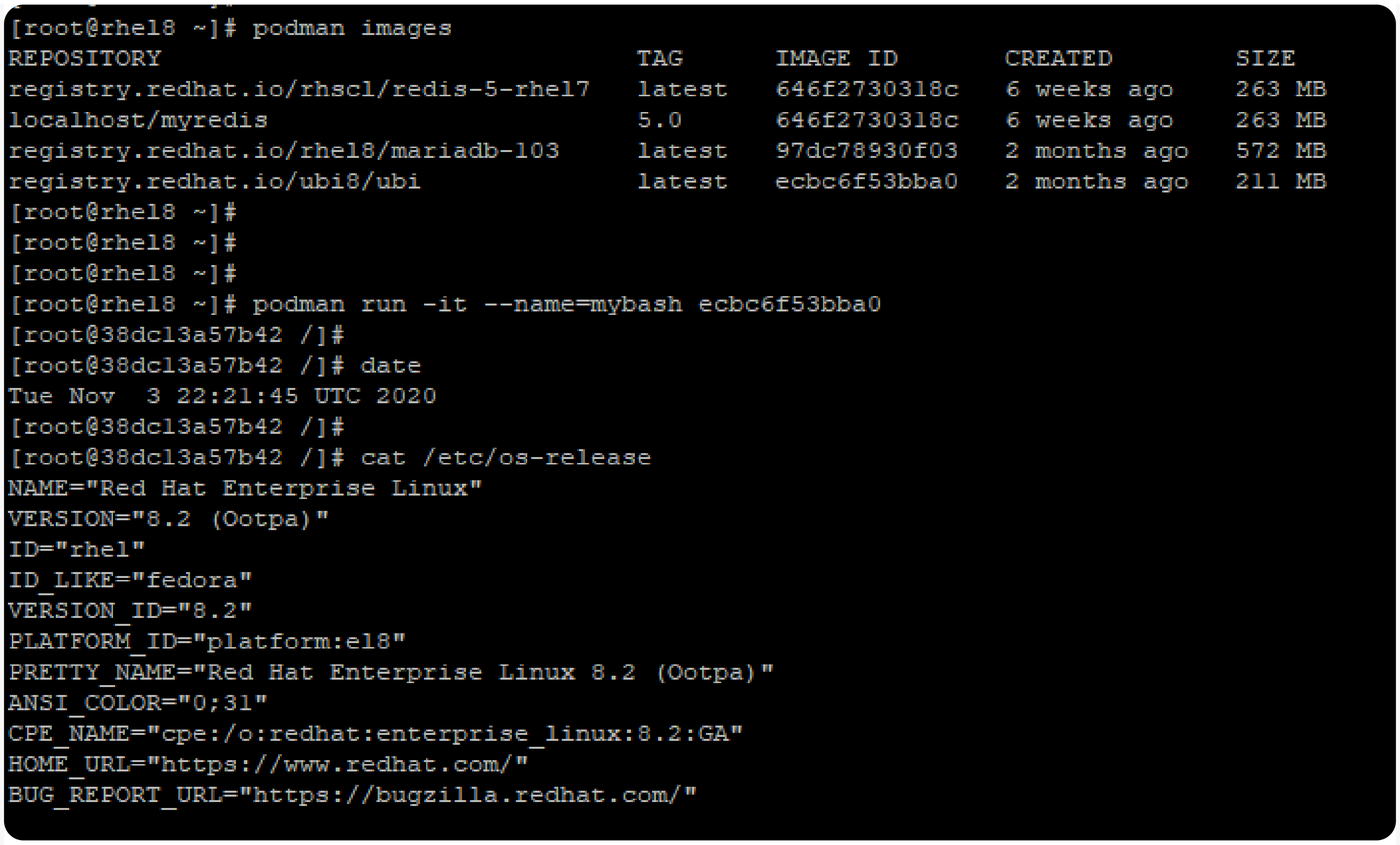
Run Redis Container Images

If you are running a container based on an operating system such as **RHEL 8** base image, you can gain access to the shell using the `-it` directive. The `-i` option creates an interactive session while the `-t` spawns a terminal session. The `--name` option sets the container name to **mybash** while is the **ecbc6f53bba0** image id of the base image.

```
# podman run -it --name=mybash ecbc6f53bba0
```

Thereafter, you can run any shell commands. In the example below, we are verifying the OS version of the container image.

```
# cat /etc/os-release
```



```
[root@rhel8 ~]# podman images
REPOSITORY                              TAG      IMAGE ID       CREATED        SIZE
registry.redhat.io/rhscl/redis-5-rhel7  latest   646f2730318c   6 weeks ago    263 MB
localhost/myredis                       5.0      646f2730318c   6 weeks ago    263 MB
registry.redhat.io/rhel8/mariadb-103    latest   97dc78930f03   2 months ago   572 MB
registry.redhat.io/ubi8/ubi             latest   ecbc6f53bba0   2 months ago   211 MB
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# podman run -it --name=mybash ecbc6f53bba0
[root@38dc13a57b42 /]#
[root@38dc13a57b42 /]# date
Tue Nov  3 22:21:45 UTC 2020
[root@38dc13a57b42 /]#
[root@38dc13a57b42 /]# cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.2 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.2"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.2 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8.2:GA"
HOME_URL="https://www.redhat.com/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"
```

Verify Container Image OS Version

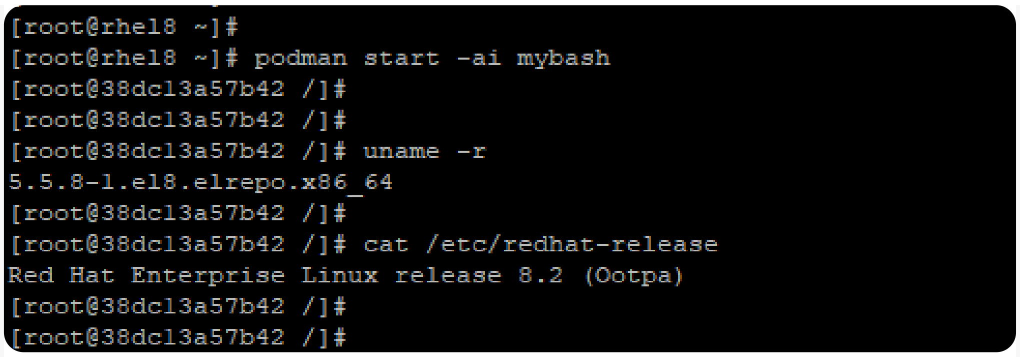
To exit the container, simply invoke the exit command.

```
# exit
```

Once the container is exited, it automatically stops. To start the container again, use the **podman start** command with the `-ai` flag as shown.

```
# podman start -ai mybash
```

Once again, this gives you access to the shell.



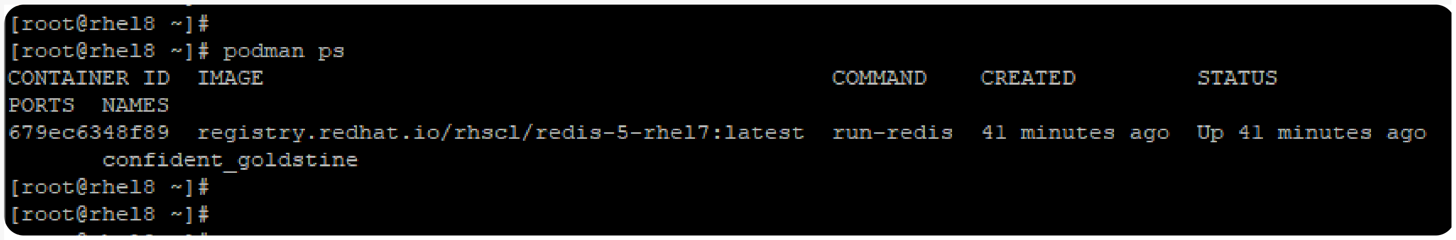
```
[root@rhel8 ~]#
[root@rhel8 ~]# podman start -ai mybash
[root@38dc13a57b42 /]#
[root@38dc13a57b42 /]#
[root@38dc13a57b42 /]# uname -r
5.5.8-1.el8.elrepo.x86_64
[root@38dc13a57b42 /]#
[root@38dc13a57b42 /]# cat /etc/redhat-release
Red Hat Enterprise Linux release 8.2 (Ootpa)
[root@38dc13a57b42 /]#
[root@38dc13a57b42 /]#
```

Start Container Images

Listing Running Container Images

To list currently running containers, use the **podman ps** command as shown.

```
# podman ps
```

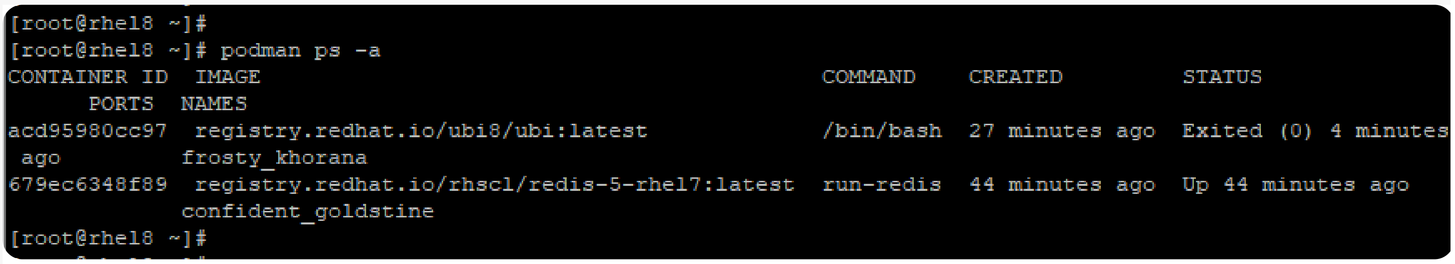


```
[root@rhel8 ~]#
[root@rhel8 ~]# podman ps
CONTAINER ID  IMAGE                                          COMMAND    CREATED         STATUS
PORTS  NAMES
679ec6348f89  registry.redhat.io/rhscl/redis-5-rhel7:latest  run-redis  41 minutes ago  Up 41 minutes ago
       confident_goldstine
[root@rhel8 ~]#
[root@rhel8 ~]#
```

List Running Container Images

To view all containers including those ones that have exited after running, use the command:

```
# podman ps -a
```



```
[root@rhel8 ~]#
[root@rhel8 ~]# podman ps -a
CONTAINER ID  IMAGE                                          COMMAND    CREATED         STATUS
      PORTS  NAMES
acd95980cc97  registry.redhat.io/ubi8/ubi:latest             /bin/bash  27 minutes ago  Exited (0) 4 minutes
 ago         frosty_khorana
679ec6348f89  registry.redhat.io/rhscl/redis-5-rhel7:latest  run-redis  44 minutes ago  Up 44 minutes ago
             confident_goldstine
[root@rhel8 ~]#
```

View All Container Images

Configure Container Images to Auto Start Under Systemd Service

In this section, we focus on how a container can be configured to run directly on an RHEL system as a systemd service.

First, get your preferred image. In this case, we have pulled the **Redis** image from docker hub:

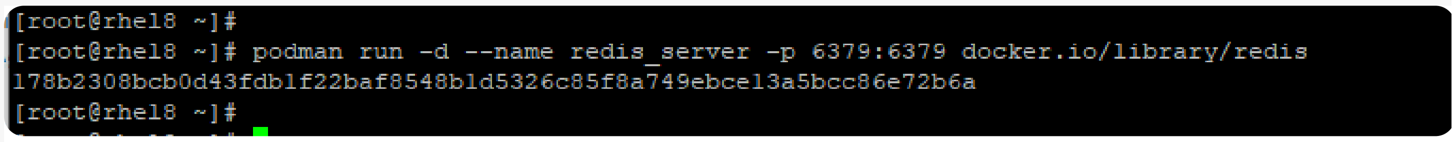
```
# podman pull docker.io/redis
```

If you have SELinux running on your system, you need to activate the **container_manage_cgroup** boolean to run containers with **systemd**.

```
# setsebool -p container_manage_cgroup on
```

Thereafter, run the container image in the background and assign it to your preferred image name. In this example, we have named our image **redis_server** and mapped the port **6379** from the container to our **RHEL 8** host

```
# podman run -d --name redis_server -p 6379:6379 redis
```



```
[root@rhel8 ~]#
[root@rhel8 ~]# podman run -d --name redis_server -p 6379:6379 docker.io/library/redis
178b2308bcb0d43fdb1f22baf8548b1d5326c85f8a749ebce13a5bcc86e72b6a
[root@rhel8 ~]#
```

Run Container Image in Background

Next, we are going to create a **systemd** unit configuration file for redis in the **/etc/systemd/system/** directory.

```
# vim /etc/systemd/system/redis-container.service
```

Paste the content below to the file.

```
[Unit]
Description=Redis container

[Service]
Restart=always
ExecStart=/usr/bin/podman start -a redis_server
ExecStop=/usr/bin/podman stop -t 2 redis_server

[Install]
WantedBy=local.target
```

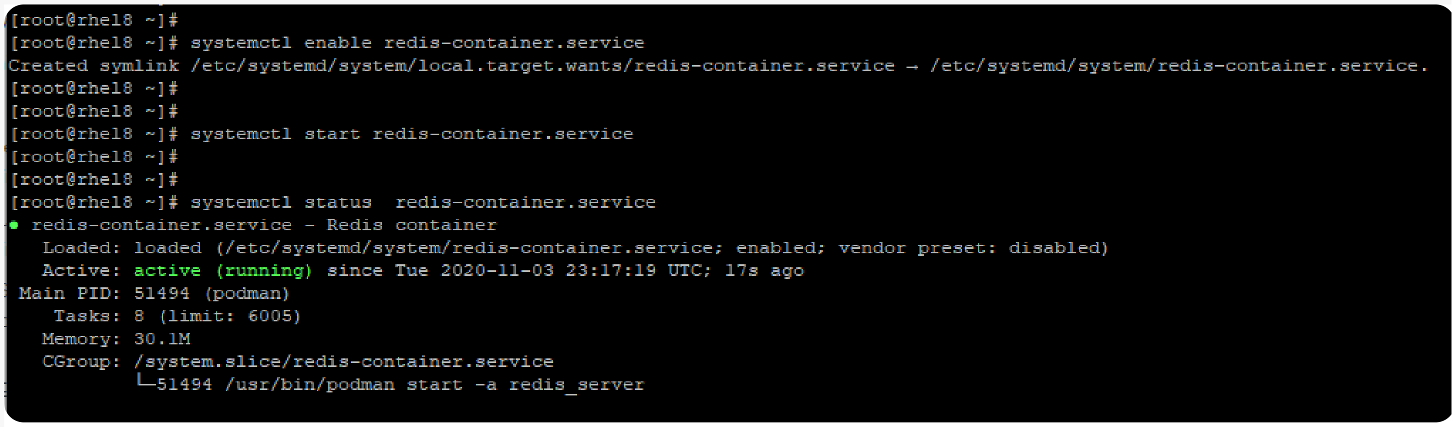
Save and exit the file.

Next, configure the container to start automatically on bootup.

```
# systemctl enable redis-container.service
```

Next, start the container and verify its running status.

```
# systemctl start redis-container.service
# systemctl status redis-container.service
```



```
[root@rhel8 ~]#
[root@rhel8 ~]# systemctl enable redis-container.service
Created symlink /etc/systemd/system/local.target.wants/redis-container.service → /etc/systemd/system/redis-container.service.
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# systemctl start redis-container.service
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# systemctl status  redis-container.service
● redis-container.service - Redis container
   Loaded: loaded (/etc/systemd/system/redis-container.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2020-11-03 23:17:19 UTC; 17s ago
 Main PID: 51494 (podman)
    Tasks: 8 (limit: 6005)
   Memory: 30.1M
   CGroup: /system.slice/redis-container.service
           └─51494 /usr/bin/podman start -a redis_server
```

Verify Container Image Status

Configure Persistent Storage for Container Images

When running containers, it's prudent to configure persistent external storage on the host. This provides a backup in case the container crashes or gets removed accidentally.

To persist the data, we are going to map a directory located in the host to a directory inside the container.

```
$ podman run --privileged -it -v /var/lib/containers/backup_storage:/
```

The `--privileged` option is passed when **SELinux** is set to enforcing. The `-v` option specifies the external volume which is located on the host. The container volume here is the **/mnt** directory.

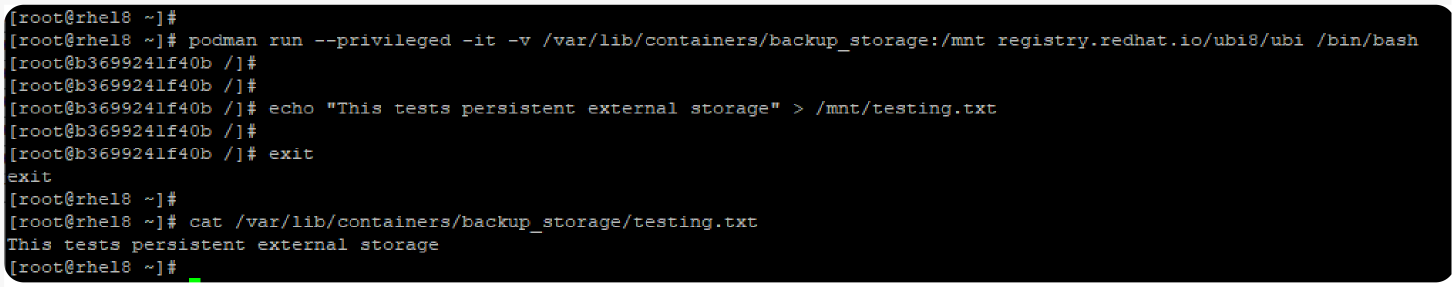
Once we have accessed the shell, we are going to create a sample file **testing.txt** in the **/mnt** directory as shown.

```
$ echo "This tests persistent external storage" > /mnt/testing.txt
```

We will then **exit** the container and check whether the file exists in the external storage residing on the host

```
# exit
# cat /var/lib/containers/backup_storage/testing.txt
```

Output ⇒ **This tests persistent external storage**.



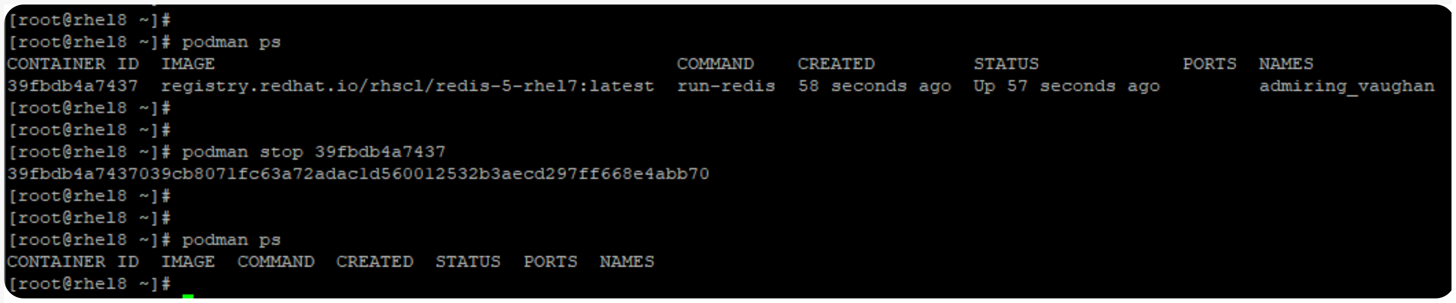
```
[root@rhel8 ~]#
[root@rhel8 ~]# podman run --privileged -it -v /var/lib/containers/backup_storage:/mnt registry.redhat.io/ubi8/ubi /bin/bash
[root@b3699241f40b /]#
[root@b3699241f40b /]#
[root@b3699241f40b /]# echo "This tests persistent external storage" > /mnt/testing.txt
[root@b3699241f40b /]#
[root@b3699241f40b /]# exit
exit
[root@rhel8 ~]#
[root@rhel8 ~]# cat /var/lib/containers/backup_storage/testing.txt
This tests persistent external storage
[root@rhel8 ~]#
```

Configure Persistent Storage for Containers

Stopping and Removing Containers

Once you are done with running your container, you can stop it using the **podman stop**command followed by the **container-id** which you can obtain from the **podman ps**command.

```
# podman stop container-id
```



```
[root@rhel8 ~]#
[root@rhel8 ~]# podman ps
CONTAINER ID  IMAGE                                      COMMAND    CREATED         STATUS             PORTS  NAMES
39fbdb4a7437  registry.redhat.io/rhscl/redis-5-rhel7:latest  run-redis  58 seconds ago  Up 57 seconds ago         admiring_vaughan
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# podman stop 39fbdb4a7437
39fbdb4a7437039cb8071fc63a72adac1d560012532b3aecd297ff668e4abb70
[root@rhel8 ~]#
[root@rhel8 ~]#
[root@rhel8 ~]# podman ps
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES
[root@rhel8 ~]#
```

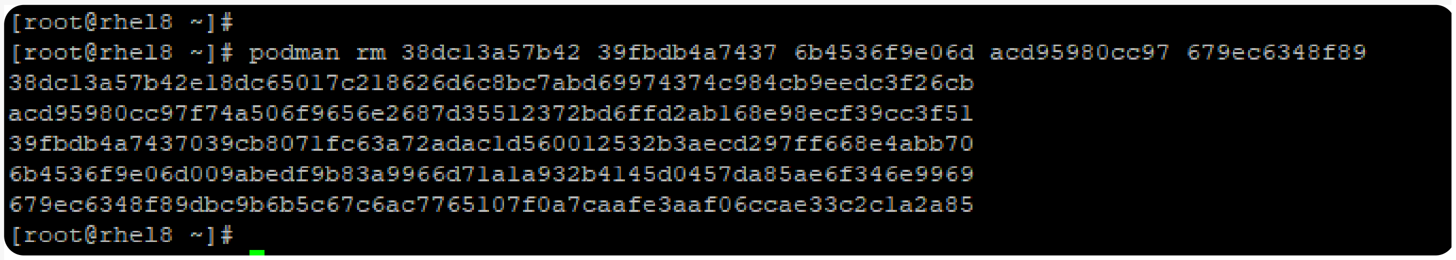
Stop Container Image

To remove the containers that you no longer need, first, ensure that you stop it and then invoke the **podman rm** command followed by the container id or name as an option.

```
# podman rm container-id
```

To remove multiple containers at a go in one command, specify the container **ids**separated by a space.

```
# podman rm container-id-1 container-id-2 container-id-3
```

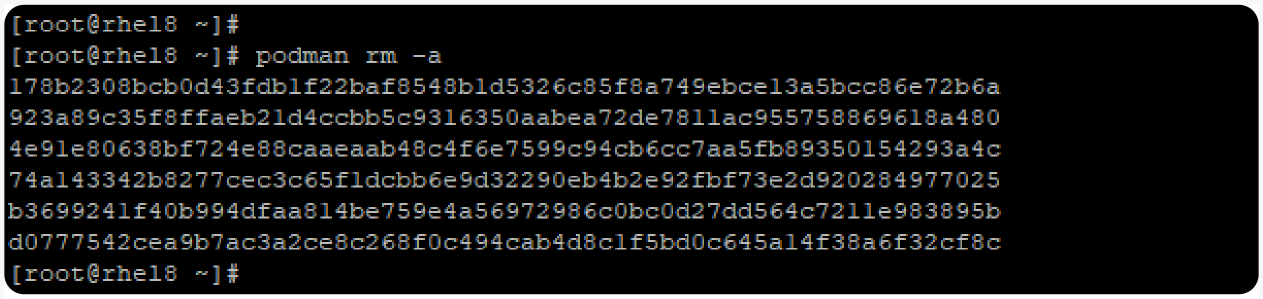


```
[root@rhel8 ~]#
[root@rhel8 ~]# podman rm 38dc13a57b42 39fbdb4a7437 6b4536f9e06d acd95980cc97 679ec6348f89
38dc13a57b42e18dc65017c218626d6c8bc7abd69974374c984cb9eedc3f26cb
acd95980cc97f74a506f9656e2687d35512372bd6ffd2ab168e98ecf39cc3f51
39fbdb4a7437039cb8071fc63a72adac1d560012532b3aecd297ff668e4abb70
6b4536f9e06d009abedf9b83a9966d71a1a932b4145d0457da85ae6f346e9969
679ec6348f89dbc9b6b5c67c6ac7765107f0a7caafe3aaf06ccae33c2c1a2a85
[root@rhel8 ~]#
```

Remove Container Image

To clear all your containers, run the command:

```
# podman rm -a
```



```
[root@rhel8 ~]#
[root@rhel8 ~]# podman rm -a
178b2308bcb0d43fdb1f22baf8548b1d5326c85f8a749ebce13a5bcc86e72b6a
923a89c35f8ffaeb21d4ccbb5c9316350aabea72de7811ac955758869618a480
4e91e80638bf724e88caaeaab48c4f6e7599c94cb6cc7aa5fb89350154293a4c
74a143342b8277cec3c65f1dcbb6e9d32290eb4b2e92fbf73e2d920284977025
b3699241f40b994dfaa814be759e4a56972986c0bc0d27dd564c7211e983895b
d0777542cea9b7ac3a2ce8c268f0c494cab4d8c1f5bd0c645a14f38a6f32cf8c
[root@rhel8 ~]#
```

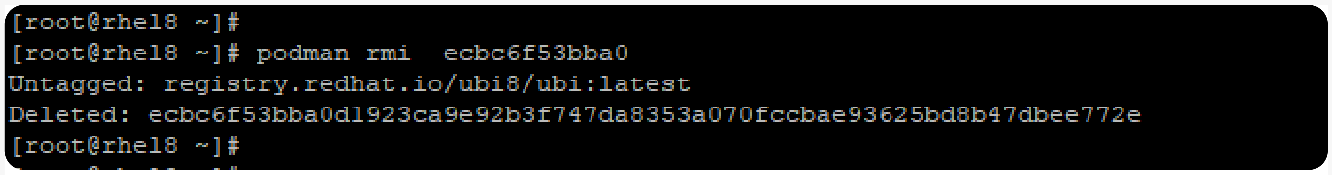
Clear All Containers

Removing an Image

To remove an image, first, ensure that all containers spawned from the images are stopped and removed as discussed in the previous sub-topic.

Next, proceed and run the podman `-rmi` command followed by the ID of the image as shown:

```
# podman -rmi image-id
```



```
[root@rhel8 ~]#
[root@rhel8 ~]# podman rmi  ecbc6f53bba0
Untagged: registry.redhat.io/ubi8/ubi:latest
Deleted: ecbc6f53bba0d1923ca9e92b3f747da8353a070fccbae93625bd8b47dbee772e
[root@rhel8 ~]#
```

Remove Image

Conclusion

This wraps up this chapter on managing and working with containers in **RHEL 8**. We hope this guide provided a decent understanding of containers and how you can interact and manage them on your **RHEL** system using **podman** and **Skopeo**.