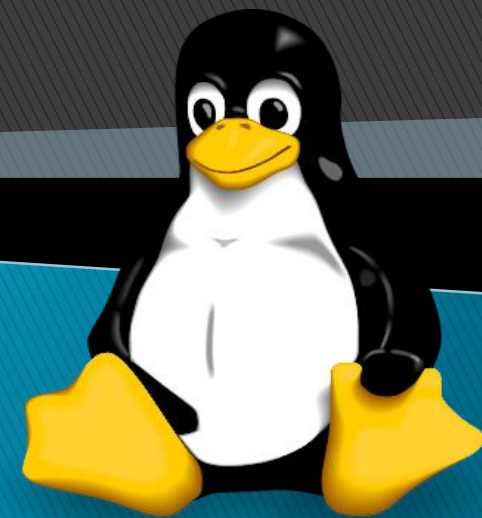
Running Container



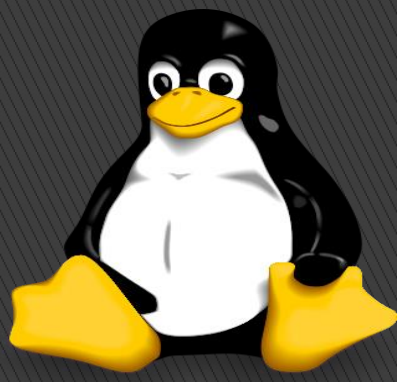
CONTAINERIZATION

Over the years, technology has revolutionized processes and paved the way for more efficient, innovative, and user-friendly software development.

Introduction of Container

Linux containers can be thought of as a lightweight alternative to virtualization. In a virtualized environment, a virtual machine is created that contains and runs the entire guest operating system. The virtual machine, in turn, runs on top of an environment such as a hypervisor that manages access to the physical resources of the host system.

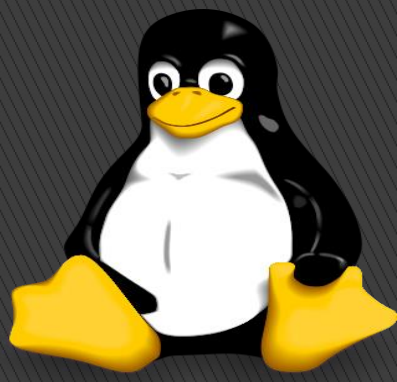
Containers work by using a concept referred to as kernel sharing which takes advantage of the architectural design of Linux and UNIX-based operating systems.



Why Container

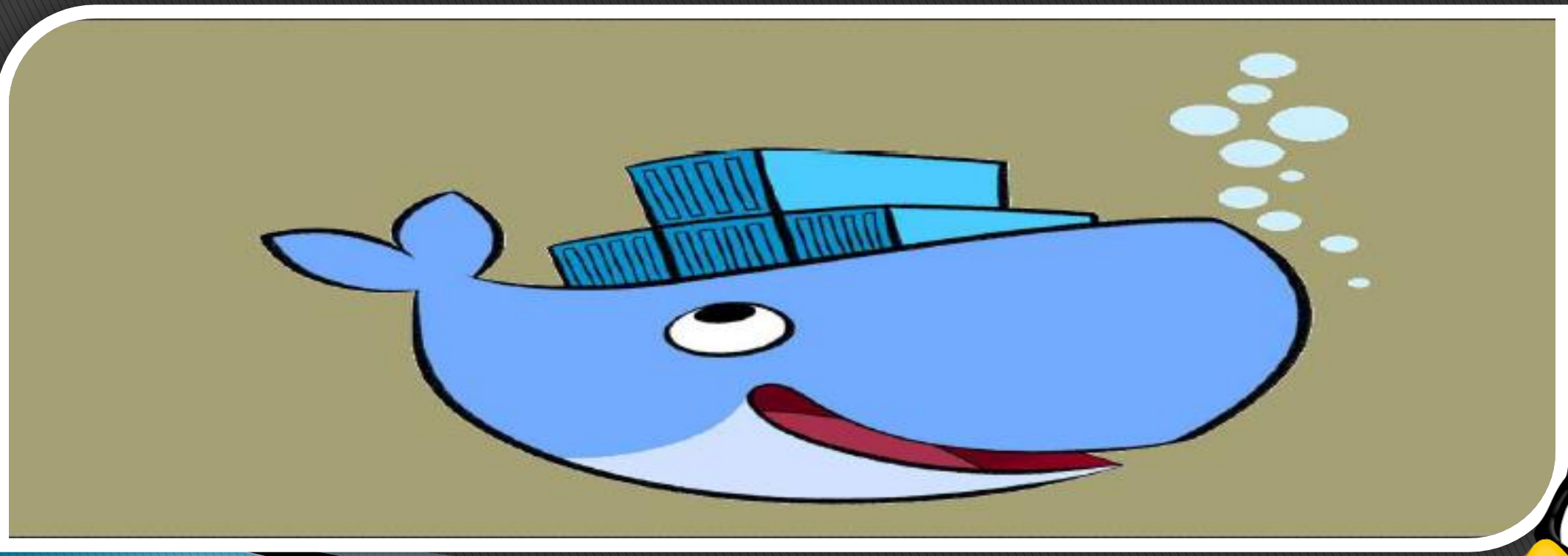
The main advantage of containers is that they require considerably less resource overhead than virtualization allowing many container instances to be run simultaneously on a single server, and can be started and stopped rapidly and efficiently in response to demand levels. Containers run natively on the host system providing a level of performance that cannot be matched by a virtual machine.

Containers are also extremely portable and can be migrated between systems quickly and easily. When combined with a container management system such as Docker, OpenShift and Kubernetes, it is possible to deploy and manage containers on a vast scale spanning multiple servers and cloud platforms, potentially running thousands of containers.



Need of Container

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

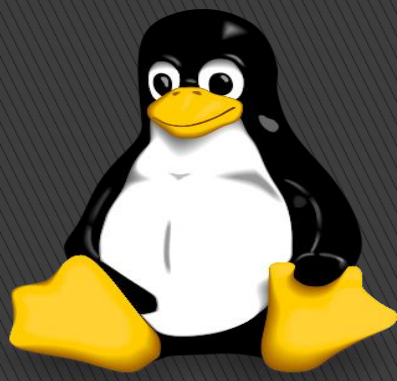


Virtualization Vs Containerization

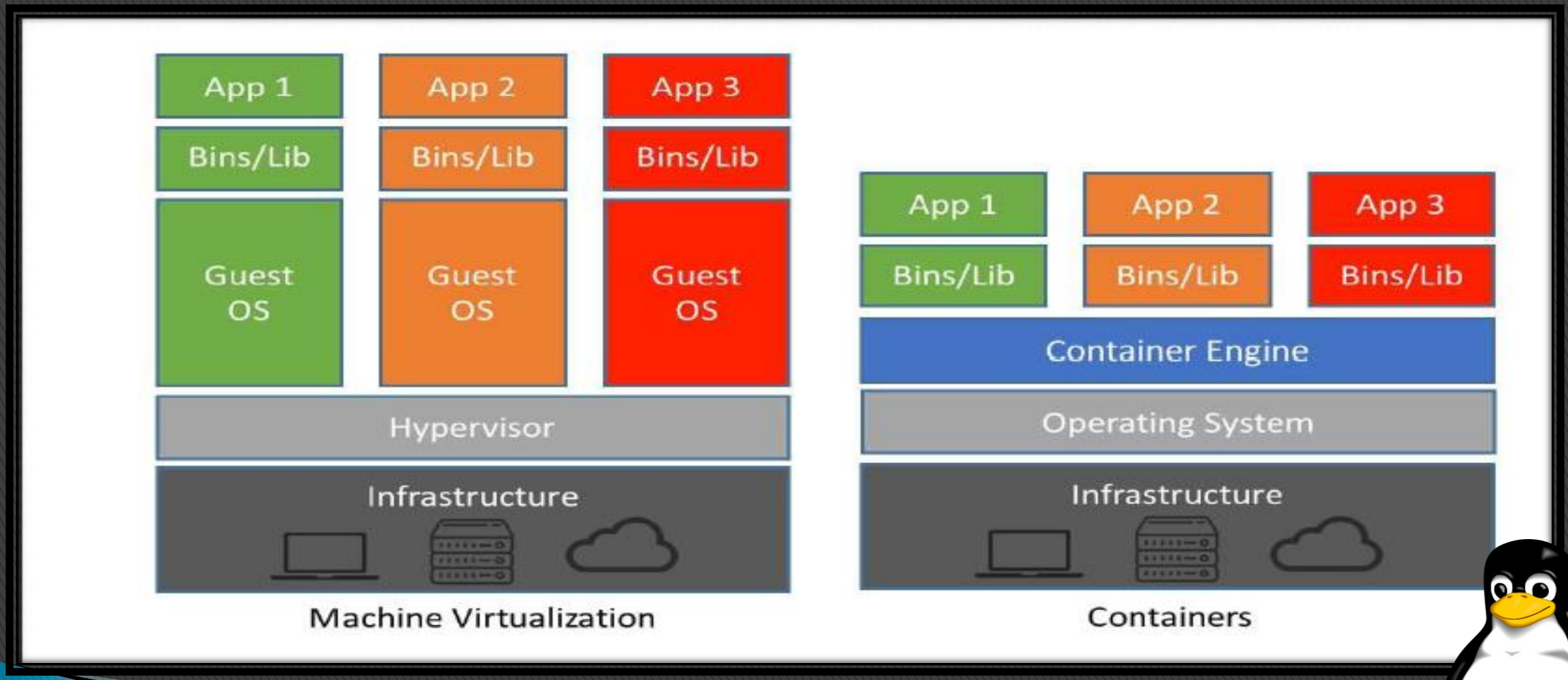
Run directly on the operating system, sharing hardware and OS resources across all containers on the system. This enables applications to stay lightweight and run swiftly in parallel.

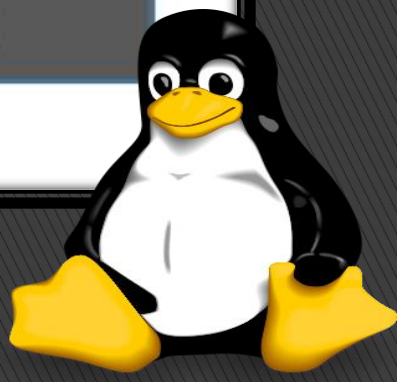
Share the same operating system kernel, isolate the containerized application processes from the rest of the system, and use any software compatible with that kernel.

Require far fewer hardware resources than virtual machines, which also makes them quick to start and stop and reduces storage requirements.



Virtualization Vs Containerization





Virtualization Vs Containerization

Virtual Machine

1. Heavyweight
2. Limited performance
3. Each VM run in its own OS
4. Hardware level virtualization
5. Startup time in minutes
6. Allocate Required memory
7. Fully isolated

Container

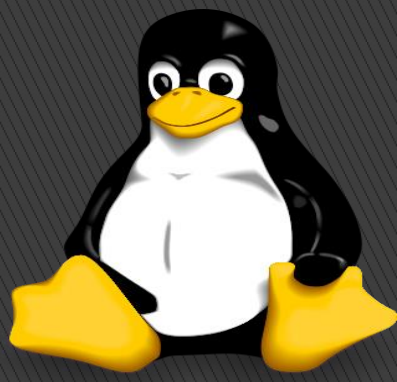
1. Lightweight
2. Native performance
3. All container share host OS
4. OS level Virtualization
5. Startup time in milliseconds
6. Require less memory
7. Process level isolation

Comparing Containers to Virtual Machines

Containers provide many of the same benefits as virtual machines, such as security, storage, and network isolation.

Both technologies isolate their application libraries and runtime resources from the host Operating system or hypervisor and vice versa.

Containers and Virtual Machines are different in the way they interact with hardware and the underlying operating system.



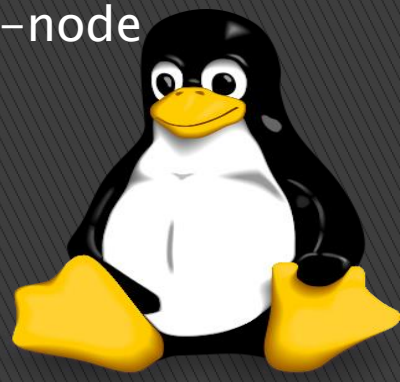
Managing Containers with Podman

A good way to start learning about containers is to work with individual containers on a single server acting as a container host. Red Hat Enterprise Linux provides a set of container tools that

you can use to do this, including:

- podman, which directly manages containers and container images.
- skopeo, which you can use to inspect, copy, delete, and sign images.
- buildah, which you can use to create new container images.

These tools are compatible with the Open Container Initiative (OCI). They can be used to manage any Linux containers created by OCI-compatible container engines, such as Docker. These tools are specifically designed to run containers under Red Hat Enterprise Linux on a single-node container host.

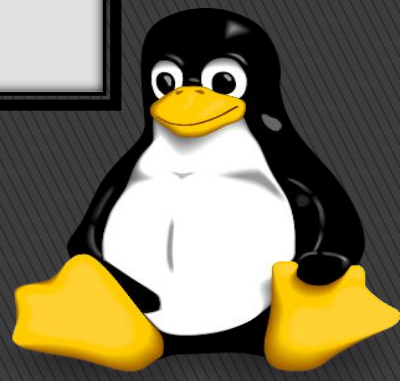


PODMAN

Getting Started with

podman

On RHEL 8



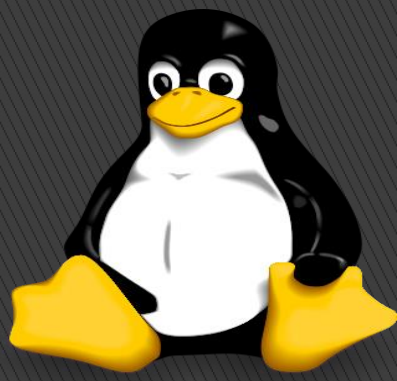
What is podman

podman – for directly managing pods and container images (run, stop, start, ps, attach, exec, and so on)

A Pod (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage and network resources, and a specification

Podman is similar to Docker and has the same command options except that Docker is adaemon.

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.



Container Naming Conventions

Container images are named based on the following fully qualified image name syntax:

registry_name/user_name/image_name:tag

- The registry_name is the name of the registry storing the image. It is usually the fully qualified domain name of the registry.

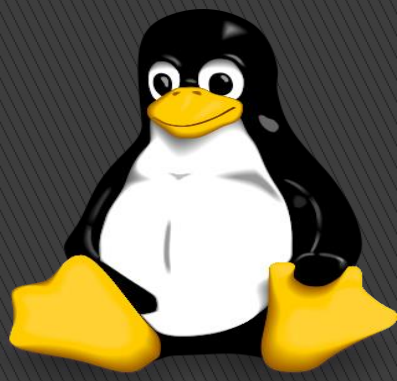
- The user_name represents the user or organization to which the image belongs.

- The image_name must be unique in the user namespace.

- The tag identifies the image version. If the image name includes no image tag, then latest is assumed.

For check podman installed or not

#rpm -q podman
#which podman
#whereis podman
#yum info podman



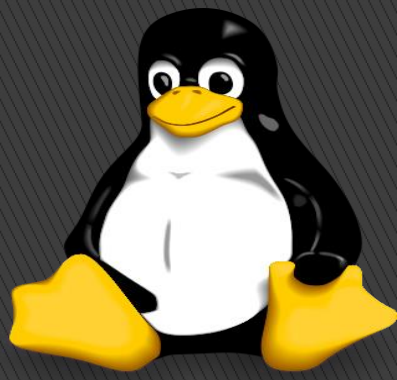
Install podman

for install podman
#yum install podman

for install container
#yum module install container-tools -y

Check podman version
#podman -v

For check available images in system
#podman images



To Create Container

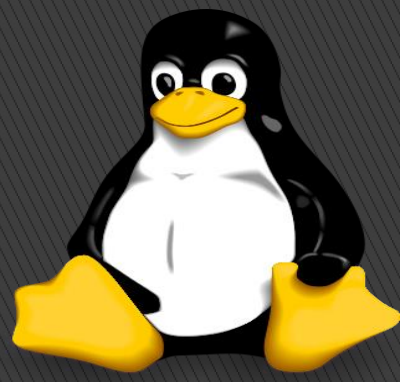
To run a container on your local system, you must first pull a container image. Use Podman to pull an image from a registry. You should always use the fully qualified image name when pulling images.

The podman pull command pulls the image you specify from the registry and saves it locally:

#podman pull registry.access.redhat.com/ubi8/ubi:latest

After retrieval, Podman stores images locally and you can list them using the podman images

#podman images



To run a container

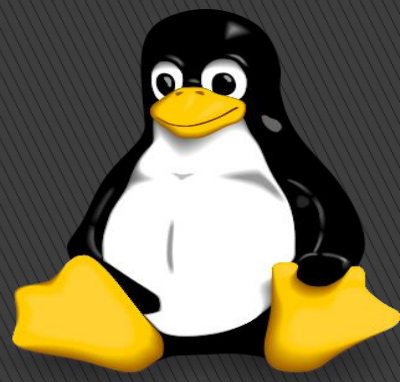
To run a container from this image, use the podman run command.

#podman run -it registry.access.redhat.com/ubi8/ubi:latest
#exit

Use the -it options to interact with the container, if required. The -it options allocate a terminal to the container and allow you to send keystrokes to it.

Many Podman flags also have an alternative long form; some of these are explained below.

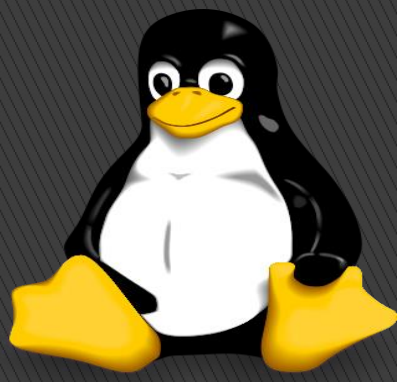
- -t is equivalent to terminal is allocated for the container.
- -i is the same as --interactive. When this option is used, the container accepts standard input.
- -d is for --detach, means the container runs in the background (detached).
- --name - it use to set name to container.



Run container with specific Name

```
#podman run -it --name=rhel8 registry.access.redhat.com/ubi8/ubi
/bin/bash
#cat  /etc/os-release
#exit

#podman ps  -a
```



Search pull and run container

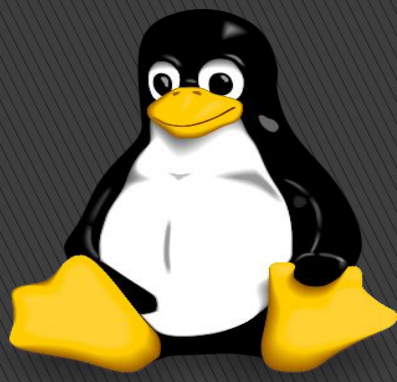
for search images
#podman search ubi

For pull images
#podman pull docker.io/redhat/ubi8

For show images
#podman images

we can directly run image without pull
#podman run -it docker.io/redhat/ubi8 /bin/bash
#exit
Note
After exit running container will be exited (stop)
For show running container
#podman ps

For show running and stop container
#podman ps -a

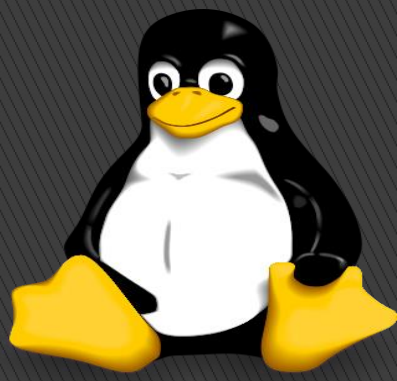


run a quick command in a container

You can also run a quick command in a container without interacting with it, and then remove the container once the command is completed. To do this, use podman run --rm followed by the container image and a command.

#podman run --rm registry.access.redhat.com/ubi8/ubi cat /etc/osrelease

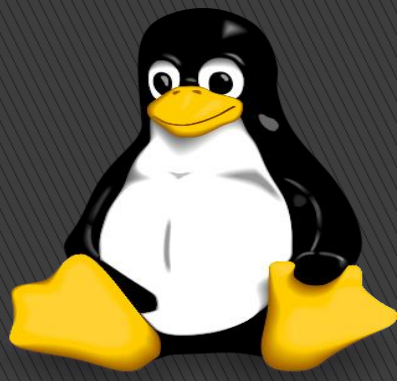
#podman ps
#podman ps -a



Registry.conf

Podman uses a registries.conf file on your host system to get information about the container registries it can use.

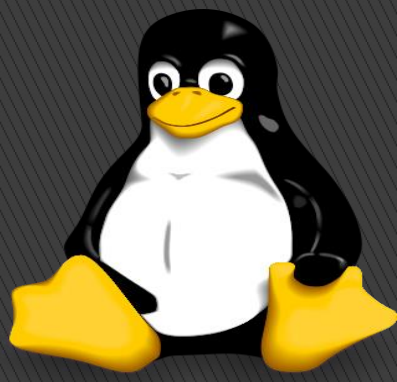
#cat /etc/containers/registries.conf



Podman info

The podman info command displays configuration information for Podman, including its configured registries.

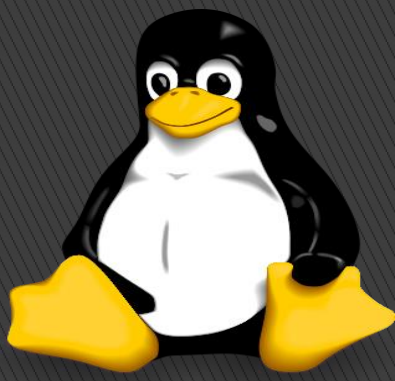
#podman info



Finding Container Images

Use the podman search command to search container registries for a specific container image.

#podman search registry.redhat.io/rhel8



Inspecting Container Images

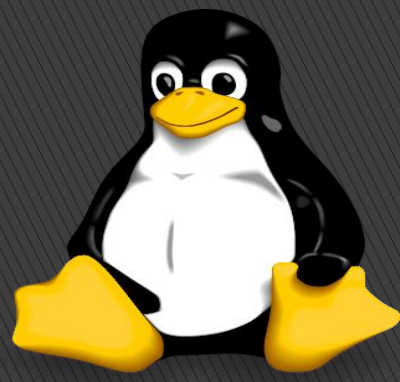
inspect locally stored image information using the podman inspect command. This command might provide more information than the skopeo inspect command.

List locally stored images:

#podman images

Inspect a locally stored image and return information:

#podman inspect registry.redhat.io/rhel8/python-36

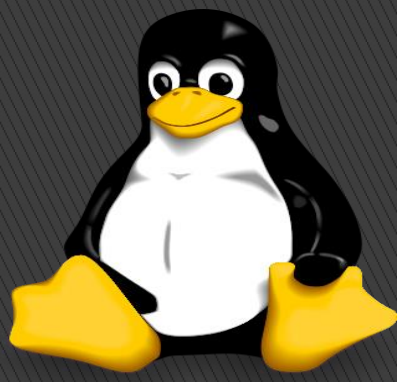


Removing Local Container Images

To remove a locally stored image, use the podman rmi command.
List locally stored images:
#podman images

Remove the registry.redhat.io/rhel8/python-36:latest image.
#podman rmi registry.redhat.io/rhel8/python-36:latest

List locally stored images and verify that it was removed:
#podman images

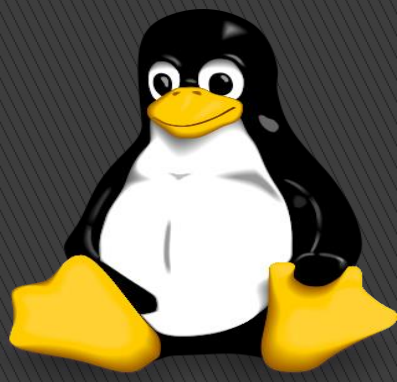


Running Commands in a Container

Running Commands in a Container

When a container starts, it executes the container image's entry point command. However, you might need to execute other commands to manage the running container.

#podman exec 7ed6e671a600 cat /etc/redhat-release



Start stop kill container

stop container
#podman stop my-httpd-container

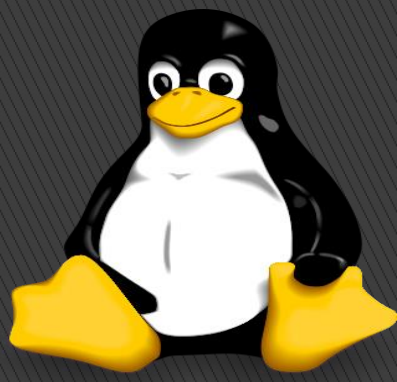
start container
#podman start my-httpd-container

Restart container
#podman restart my-httpd-container

Kill container
#podman kill my-httpd-container

Inspect Container
#podman inspect my-httpd-container

Info Container
#podman info my-httpd-container

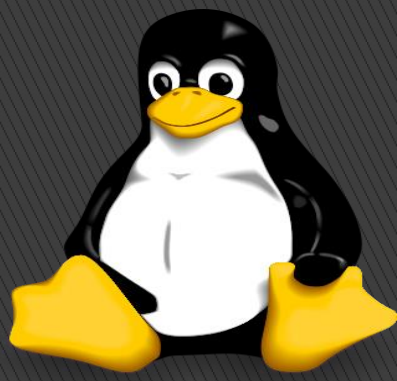


Rootfull and Rootless Containers

If container create by using root user then it called rootfull.

If container create/run using normal user then it called rootless container.

when we start container with root then we got full privilege but it not good if attacker access system, so expert recommended run container in rootless mode.



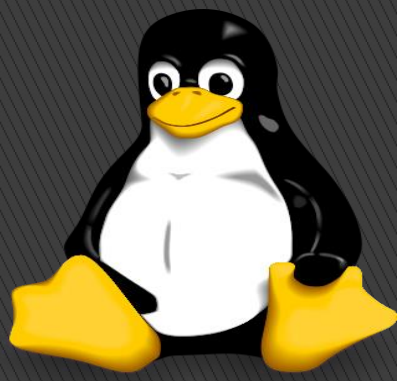
Config web server in container

For check available images in system
#podman images

for search images
#podman search httpd

For pull images
#podman pull docker.io/library/httpd

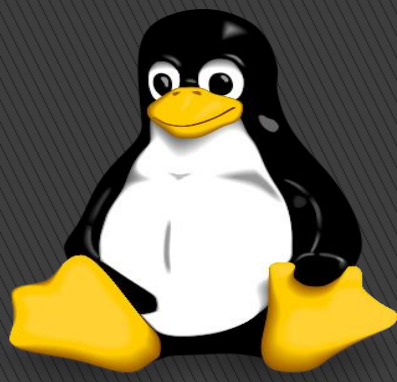
For show images
#podman images



Run web server

we can directly run image without pull
#podman run -dt -t -p 8080:80 --name=myweb docker.io/library/httpd

For check test web page (or use web browser)
#curl http://localhost:8080



Modify webpage running inside container

For show running container
#podman ps

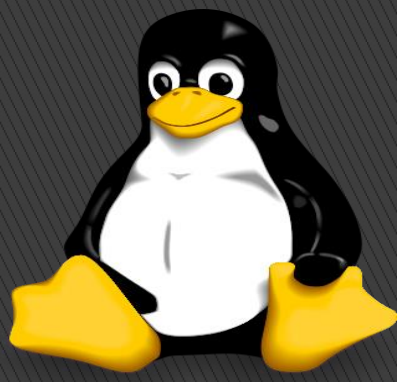
#podman exec -it myweb /bin/bash
#cd htdocs
#cat index.html

#cat > index.html
<html><head><title>MyWeb</title></head>
<body bgcolor=skyblue>
<h1>Welcome to Apache WebServer Hosted Insite Container</h1>
</body>
</html>

Ctrl+d (save & quit)

#exit (exit from container)

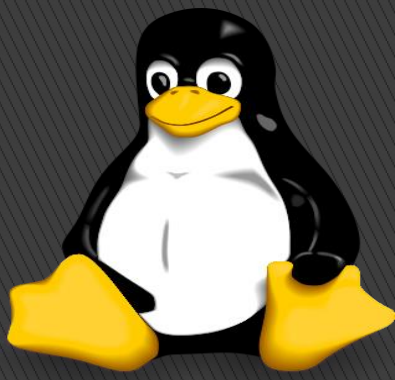
For check test web page (or use web browser)
#curl http://localhost:8080



For show running and stop container

#podman ps -a

#podman stop -a
#podman rm -a



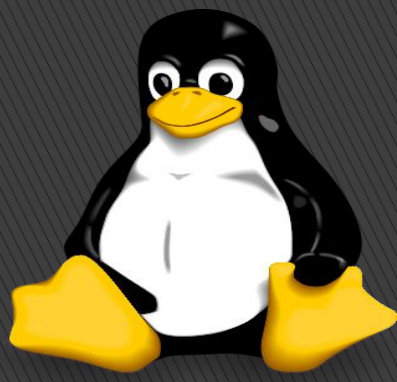
Now create directory to share web page with container

#mkdir /html
#echo "welcome to SeveMentor Pune" > /html/index.html

#chmod 777 /html/index.html
#ls -ldZ /html
#chcon -t httpd_sys_content_t /html
#chcon -t httpd_sys_content_t /html/index.html

For start container
#podman run -dt -t -p 8080:80 --name=myweb -v /html/:/usr/local/apache2/htdocs/ docker.io/library/httpd
#podman ps

For check test web page (or use web browser)
#curl http://localhost:8080



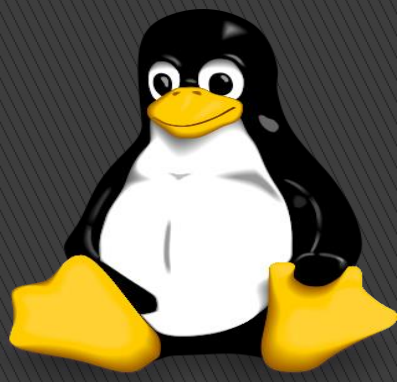
Host nginx web server

#podman pull nginx

.

.

same like above

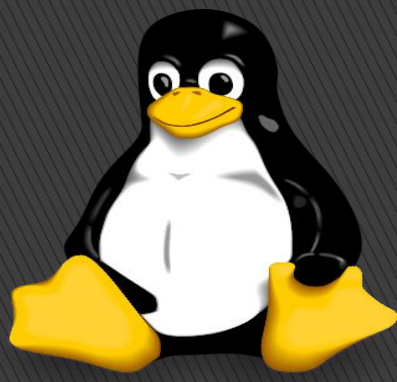


Host mysql in container

```
#podman serach mysql

#podman pull docker.io/library/mysql
#podman images
#podman run -dt -t -p 3306:3306 --name=mysql -e MYSQL_ROOT_PASSWORD='Sql@#123db'
   docker.io/library/mysql

#podman exec -it mysql  /bin/bash
#mysql -u root -p
> show databases;
> create database mydb;
> show databases;
> exit
#exit
#podman stop -a
#podman rm -a
#podman ps -a
```



Run rootless container

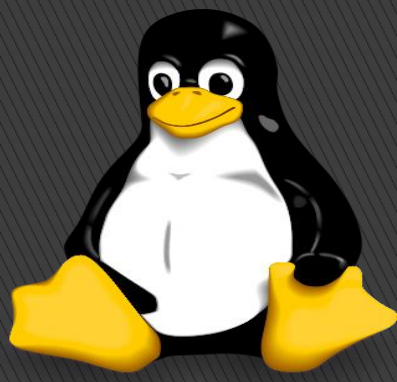
Now we try to run rootless container

#useradd rohit
#passwd rohit
#su rohit

$podman serach myubi
$podman pull docker.io/myage/myubi
$podman images

For show images details using inspect
$podman inspect docker.io/library/mysql

For check where store images
$podman info docker.io/library/mysql

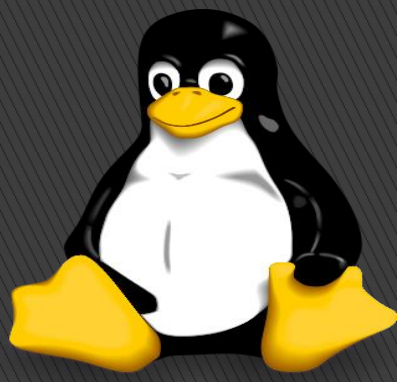


Run images rootless

Run images rootless

$podman run -it name=myubi docker.io/library/mysql

#cat /etc/os-release
#exit



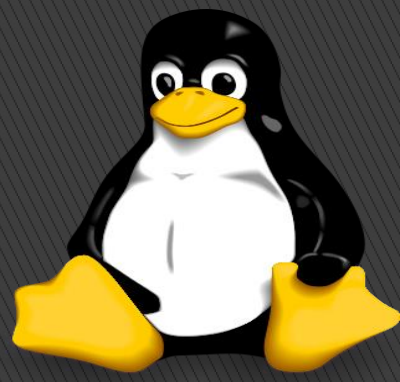
Run web server image rootless

for search images
$podman search httpd

For pull images
#podman pull docker.io/library/httpd

For show images
$podman images

we can directly run image without pull
$podman run -dt -t -p 800:80 --name=myweb docker.io/library/httpd
Note:
In rootless mode we cant use port under 1024 its limit for rootless containers.



Run web server image rootless

for search images
$podman search httpd

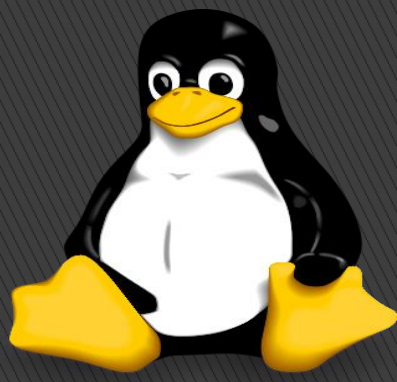
For pull images
#podman pull docker.io/library/httpd

For show images
$podman images

we can directly run image without pull
$podman run -dt -t -p 800:80 --name=myweb docker.io/library/httpd
Note:
In rootless mode we cant use port under 1024 its limit for rootless containers.

$podman run -dt -t -p 4444:80 --name=myweb docker.io/library/httpd

$curl http://localhost:4444



Red Hat Ansible Automation Platform