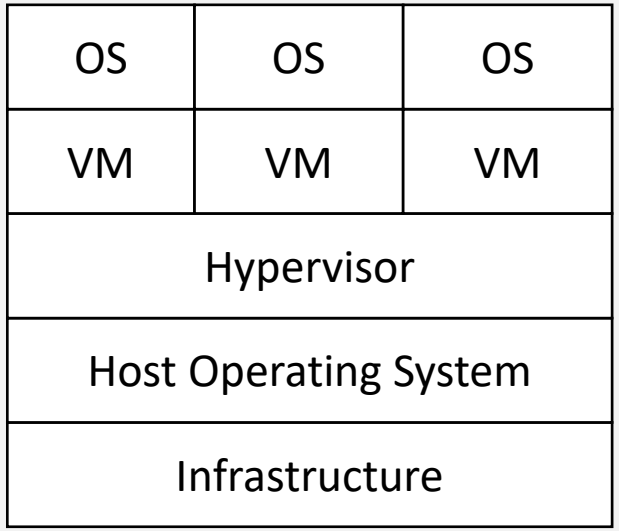
Managing Containers

- Introducing containers and container images
- Understanding difference between containers and virtual machines
- Installing container tools and getting help
- Searching and retrieving container images from a remote registry
- Inspect container images
- Perform container management using commands such as podman and skopeo
- Build a container from a Containerfile
- Perform basic container management such as running, starting, stopping, and listing running containers
- Run a service inside a container
- Configure a container to start automatically as a systemd service
- Attach persistent storage to a container

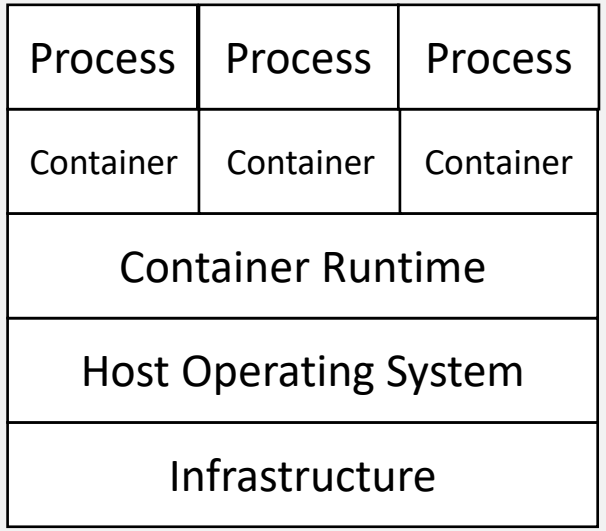
Introducing containers and container images

- Container is running code (image) in its own isolated environment. All binaries, libraries ,configs & software tools (in general resources) needed to run container are provided through image called container image.
- Container is launched from container image and can not exist without image.

- Virtual machines are example of virtualization at Infra level, but containers are example of virtualization at OS level. Different containers running on host share the same operating system but VM's don't.



Layered Architecture (VMs)



Layered Architecture (Containers)

- Docker Engine is example of popular container engine and containers launched by Docker Engine are usually called Docker containers.

- RedHat does not support Docker since RHEL-8 and built its own daemon less tools to manage individual containers, usually called Linux containers(LXC).

- Also, RedHat developed OpenShift Platform (PaaS) to run and manage (Orchestrate) containerized workload.

- Red Hat Enterprise Linux implements Linux Containers using core technologies such as Control Groups (Cgroups) for Resource Management, Namespaces (PID namespace, Network Namespace etc..) for Process Isolation.

- For RHCSA exam, You are expected to know how to manage individual Linux containers using set of tools and you don't need to know about Docker Engine and OpenShift Platform.

- podman , buildah, skopeo and runc are tools developed by RedHat to manage individual containers. We will mainly discuss about podman ,buildah and skopeo which you are expected to know for the exam.

  podman : Pod Manager for managing pods and container images.
  buildah : To build image from containerfile .
  skopeo : For copying, inspecting, deleting and signing images(remote registry operations) .

Installing Container tools and Getting help

To work with LXC's , We need to install container tools with below command.

dnf install -y container-tools

After installing container tools, we will go through podman --help, buildah --help and skopeo --help.
For more information , we can check man pages for these commands.

Introducing Rootful and Rootless containers

We can manage containers as root user (rootful containers) as well as regular users(rootless containers). Rootless containers are possible from RHEL 8.1.

So, containers started as root (privileged user)user are rootful containers and this is best way to run containers to have access to all features on the system but running containers as root is not secure and can potentially harm the host (system).

On the flip side , Containers started as regular user are rootless containers and have limited access to underlying host .Such containers are more secure, but they don't have access to all features on the system. If needed, we need to configure additional access required for those containers.

Networking is different for rootful and rootless containers , So, different networks modes are used for rootful and rootless containers . In next lecture we will talk about default network mode for rootful and rootless containers.

Introducing default podman networking

Default Network mode for Rootful containers:

With rootful containers ,default network mode is bridge network. Bridge network has default subnet and default gateway. Each rootful container gets an IP address from this subnet which enables containers to communicate with each other and to external world.

Default Network mode for Rootless containers:

In case of rootless containers , Default network mode is slirp4netns which sets up user mode networking automatically in unprivileged network namespace.

Rootless containers do not get IP address. In user mode networking , TAP device is created (by slirp4netns program) in container's network namespace and slirp4netns program forwards the traffic to remote destinations. Rootless containers are completely isolated from each other.

So, to enable communication, container's port is exposed to host and then using host local network ,container can be accessed.

Searching and Retrieving Container images

Container images are stored at some shared central location known as container registry .

What are Container Registries ?

A container registry is a collection of repositories made to store container images which is normally located on some remote server.

Image Naming Structure:

REGISTRY_FQDN[:PORT]/USER or NAMEPACE/REPO[:TAG]
Example : registry.redhat.io/ubi9/httpd-24:latest

To search container images in different container registries , podman search command line is used and to pull the image to your machine ,podman pull command is used.

But again, before we start working with podman search and podman pull to search and pull images, we must understand registries configurations in /etc/containers/registries.conf config file.

Configurations in /etc/containers/registries.conf file are system-wide search settings but as regular user you can create your own registries.conf file in your home directory $HOME/.config/containers/registries.conf to override system wide settings.

Example registries.conf file:

unqualified-search-registries = [“example1.com”, “example2.com”, “example3.com”]

[[registry]]
prefix = “example.com/foo”
insecure = false
blocked = false
location = “remote-registry-for.example.com/bar”

Image **example.com/foo**/image:latest will translate to “**remote-registry-for.example.com/bar**/image:latest”

[[registry.mirror]]
location = “example-mirror-1.local/mirrors/foo”
insecure = true

short-name-mode = “enforcing”

- Pull Apache Web Sever Container image (httpd-24) from registry "registry.redhat.io" on your System and inspect the container image.
  - Check the Exposed ports and User ID in container image configurations.
  - Authenticate using your RedHat credentials to download image.

Command	Action/Description
podman image search httpd **or** podman search httpd	To search/list images from unqualified search registries with **httpd** in name
podman login registry.redhat.io	Authenticating to registry server using Redhat portal credentials
podman image pull IMAGE_NAME Or podman pull IMAGE_NAME	Pulling/Downloading image to System
podman image list Or podman images	To list images downloaded on System
podman image inspect IMAGE_NAME Or podman inspect IMAGE_NAME	Displaying container's (image's) related information in JSON format
podman image inspect --format '{{ .Config.ExposedPorts }}' IMAGE_NAME	Listing exposed Exposed Ports
podman image inspect --format '{{ .Config.User }}' IMAGE_NAME	Listing User ID
man podman	To display man page for **podman**
podman image pull --help	Displaying help for **podman image pull**

- Run the web server container in background from image downloaded in previous task.
  - Give name myweb to this container.
  - Verify using podman ps if container is running.
  - Stop the container and verify again if container is stopped.
  - Delete the container and container image.

Command	Action/Description
podman container run -d --name myweb IMAGE_NAME or podman run -d --name myweb IMAGE_NAME	To run webserver container in **background**
podman container ps **(Or)** podman ps	Listing running containers
podman container stop myweb **(Or)** podman stop myweb	Stopping **container**
podman container rm myweb **(Or)** podman rm myweb	Deleting **container**
podman image rm IMAGE_NAME **(Or)** podman rmi IMAGE_NAME	Deleting **container image**

- Build the web server image with tag name 'myregistry.example.com/myweb/webserver:v1' from /root/Containerfile.
  - Use fedora image as Base image.
  - Create /root/index.html file with text "Webserver container is built from custom Containerfile"
  - Copy the index.html to document root path inside image.
  - Run the container with name 'my-webserver' to listen on host port 5000 and check the connection using curl.

Command	Action/Description
vim /root/index.html Webserver container is built from custom Containerfile :wq	Creating index.html file under /root directory
vim /root/Containerfile **FROM** registry.fedoraproject.org/fedora **LABEL** server=myweb **RUN** dnf install -y httpd **COPY** index.html /var/www/html **EXPOSE** 80 **CMD** ["-D", "FOREGROUND"] **ENTRYPOINT** ["/usr/sbin/httpd"] :wq	Creating Containerfile
buildah build -t myregistry.example.com/myweb/webserver:v1 .	Building container image
podman image list	Listing images
podman container run -d --name my-webserver -p 5000:80 IMAGE_NAME	Running container
podman ps	Listing running container
curl http://system.example.com:5000	Check that web server is working fine

- Pull Apache Web server image (httpd-24) and run the container with name webserver.
  - Configure webserver to display content "Welcome to container-based web server".
  - Port 3333 should be used on host machine to receive http requests.
  - Start shell in container to verify configurations.

Command	Action/Description
podman image search httpd	Searching **httpd-24** container image
podman image pull IMAGE_NAME	Pulling/downloading image to local System
podman image inspect IMAGE_NAME	Displaying **config info** contained in Image
vim /var/www/html-content/index.html Welcome to container-based web server :wq	Creating **index.html** on host and adding message to be displayed.
podman container run --name webserver -d -p 3333:EXPOSED_PORT -v /var/www/html-content:DOCUMENT_ROOT_DIR IMAGE_NAME	Running container and publishing port(s)
podman container restart webserver	Restarting container **webserver (needed if index.html is changed)**
curl http://system.example.com:3333	Verifying if webserver serves **index.html**
podman container run --help	Getting help for **podman** command line

- Configure System to start webserver container at boot as SYSTEMD service.
  - Start/enable SYSTEMD service to make sure container will start at boot.
  - Reboot System and verify if container is running as expected.

Command	Action/Description
podman generate systemd webserver	Generating **systemd unit file** for container
vim /etc/systemd/system/httpd-container.service #Paste generated systemd unit file contents here# :wq	Creating service unit file with name **httpd-container.service**
systemctl daemon-reload	**Reloading systemd** to make changes effective
systemctl start httpd-container.service	Starting **httpd-container.service** service
systemctl enable httpd-container.service	Enabling **httpd-container.service** service
systemctl status httpd-container.service	Displaying status of **httpd-container.service** service
systemctl reboot	Reboot System
systemctl status httpd-container.service	Displaying status of **httpd-container.service** service again to verify service is **Active(running).**
man systemd.service	To display man page for **systemd service units**

Inspecting images using skopeo

skopeo program is used for remote operations on registries. Below are important operations we need to know for the exam.

Inspecting container image:

We can inspect container image using skopeo without pulling image to system.

Example command: skopeo inspect docker://registry.example.com/namespace/image:tag

Copying image from one registry to other:

Example command: skopeo copy docker://registry1.example.com/ns1/image1 docker://registry2.example.com/ns2/image2

Use skopeo login to authenticate to registries.

Mounting persistent volumes inside containers

By default , Containers use ephemeral storage to store the data and when containers are deleted, data is also deleted. But we might want that data must persist after container is deleted .

Solution for this is to mount persistent volume inside container(s). We will understand how we can mount local persistent volume from host to inside containers which can be used by container(s) to store data.

Creating volume on host:

First, we need to create volume using podman volume create command line.

podman volume create PV_NAME

This volume gets mounted on host path /var/lib/containers/storage/volumes/PV_NAME/_data for rootfull containers. For rootless containers mount point is $HOME/.local/share/containers/storage/volumes/PV_NAME/_data

Mounting volume inside container:

To mount volume inside container we use -v option which we used already to mount directory inside container.

podman container run -d --name CONTAINER_NAME -v PV_NAME:DIR_PATH IMAGE_NAME

Extending Privileges for containers to Host

There is possibility to run privileged or non-privileged containers using podman. They behave and act differently on the Host.

Below are features that can be enabled from a container to host .

- Privileges : A privileged container (--privileged) runs applications on host as root user. So, you can delete files/directories mounted from host to container even if they are owned by root.

- Process Tables: A non-privileged containers can see only processes running inside container but can not see processes running on host. Privileged container (--pid=host) on the other hand can see all processes (use ps -e) running on host as well.

- Network Interfaces : By default, container has one external network interface and one loop back interface, but privileged container (--net=host) allows to access all network interfaces directly from container.

- Inter-process communications: A privileged container (--ipc=host) can use IPC facility to see information about active messages queues ,shared memory segments etc

To explore different options , check podman run --help or man podman-run.

Running Containers Using Runlabels

Some Special RedHat container images come with labels which provide pre-defined command lines in the image itself to work with those images.

To run the pre-set command defined in specific runlabel in IMAGE, we need to execute below command:

podman container runlabel <label> IMAGE

and to display command defined with specific runlabel, we need to execute below command :

podman container runlabel --display <label> IMAGE

Different Runlabels include:

- install : Sets up host system to run the container. It creates resources on host which are needed to run container.

- run : Run the container with different command line options. Normally it opens privileges to host and mount the filesystems from host to container.

- uninstall: Cleans up the host system.

Example of RedHat image which includes runlabels is rsyslog. In next task, we will run rsyslog container using runlabels.

➢ Pull rsyslog image (registry.redhat.io/rhel9/rsyslog) to your System.
- Run container as root user using runlabels defined within the container image.
- Verify working of container and create systemd unit file to start container at boot

Command	Action/Description
podman login registry.redhat.io	Authenticating to registry
podman image pull registry.redhat.io/rhel9/rsyslog	Pulling image to **System**
podman container runlabel install --display registry.redhat.io/rhel9/rsyslog	Displaying **install** runlabel
podman container runlabel install registry.redhat.io/rhel9/rsyslog	Running **install** runlabel
podman container runlabel run --display registry.redhat.io/rhel9/rsyslog	Displaying **run** runlabel
podman container runlabel run registry.redhat.io/rhel9/rsyslog	Running **run** runlabel
podman generate systemd rsyslog > /etc/systemd/system/rsyslog-container.service	Execute this command at path **/etc/systemd/system** to create unit file
systemctl enable --now rsyslog-container.service	Starting and enabling service to start container at boot

Prerequisites to run rootless containers

podman rootless containers run in private user namespace and user(s) inside container is/are mapped to users on host. This Linux feature(feature to implement user namespaces) makes containers more secure because root user inside container (for example) is mapped to some unprivileged user on host .

So, process running as root inside container runs as some unprivileged user on the host.

In case of rootfull containers ,containers are run in host namespace. Now we will talk about User Id mappings for both cases.

For rootfull podman, **Container User Id (Default) = Host User Id**

For example, if we run **htttpd** webserver container as rootfull podman and default user inside container is **1001**, so web server process would run as User Id 1001 inside container and as well as on host.

For rootless podman, Host User Id (podman User Id) is mapped to Container User Id through intermediate Id but simply we can say that podman User Id (suppose it is 1001) on host is mapped to root user(User Id 0) inside container and other User Ids inside container are mapped to subordinate User Ids for podman user defined in /etc/subuid file and same is the case for group Ids.

Container User Id (0) -> Host User Id (1001)

Container User Id (22) -> Host User Id (First Sub User Id +22 -1)

If we assume first subordinate User Id for user is 100000 , then corresponding Host user Id would be (**100000+22-1=100021**).

So, nn host same process would run as **User Id 22** inside container and on host as **User Id - 100021**.

Pre-requisites to run rootless podman are :

- Rootless podman user must have subordinate user Id and group id range defined in /etc/subuid and /etc/subgid files which is default on RHEL 9.
- slirp4netns package is installed for user mode networking which gets installed as part of containers tool.

So, We don’t need to anything extra to run rootless containers.

- Pull Mariadb Server image to your System.
  - Run mariadb container as rootless user (user rhcsa) and publish Exposed port (3306) in image.
  - Set root password for mariadb service as mysql.
  - Verify if you can login as root from local host.

Command	**Action/Description**
podman image search mariadb	Searching **mariadb** image
podman login REGISTRY_FQDN	Authenticating to pull image
podman image pull IMAGE_NAME	Downloading image to **System**
podman image inspect IMAGE_NAME	Displaying configs inside image e.g. Exposed Ports
podman unshare chown -R container_uid:container_gid /var/lib/database	Configuring required permissions on directory on host
podman container run --name mariadb -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=mysql -v /var/lib/database:/var/lib/mysql:**Z** IMAGE_NAME	Running **mariadb** Container as **user rhcsa** and publishing Exposed port.
dnf install mysql	Installing mysql (client) on **System** Host
mysql -h 127.0.0.1 -u root -p	Testing database login

Note : All commands in this task will be executed as user rhcsa except the one to install mysql (client)

- Configure System to start mariadb container at boot.
  - Create system unit file with name mariadb-container.service for same purpose.
  - You must perform this action as user rhcsa.

Command	Action/Description
mkdir -p /home/rhcsa/.config/systemd/user	Creating Directory Path
podman generate systemd mariadb > /home/rhcsa/.config/systemd/user/mariadb-container.service	Generate systemd unit file and put the contents in /home/rhcsa/.config/systemd/user/mariadb-container.service file
systemctl --user daemon-reload	Reload systemd daemon
systemctl --user start mariadb-container.service	Starting service using user instance of systemd
systemctl --user start mariadb-container.service	Enabling service using user instance of systemd
systemctl --user status mariadb-container.service	Checking status of containerized service
systemctl reboot	Rebooting system to verify if containerized service is started at boot (as root user)