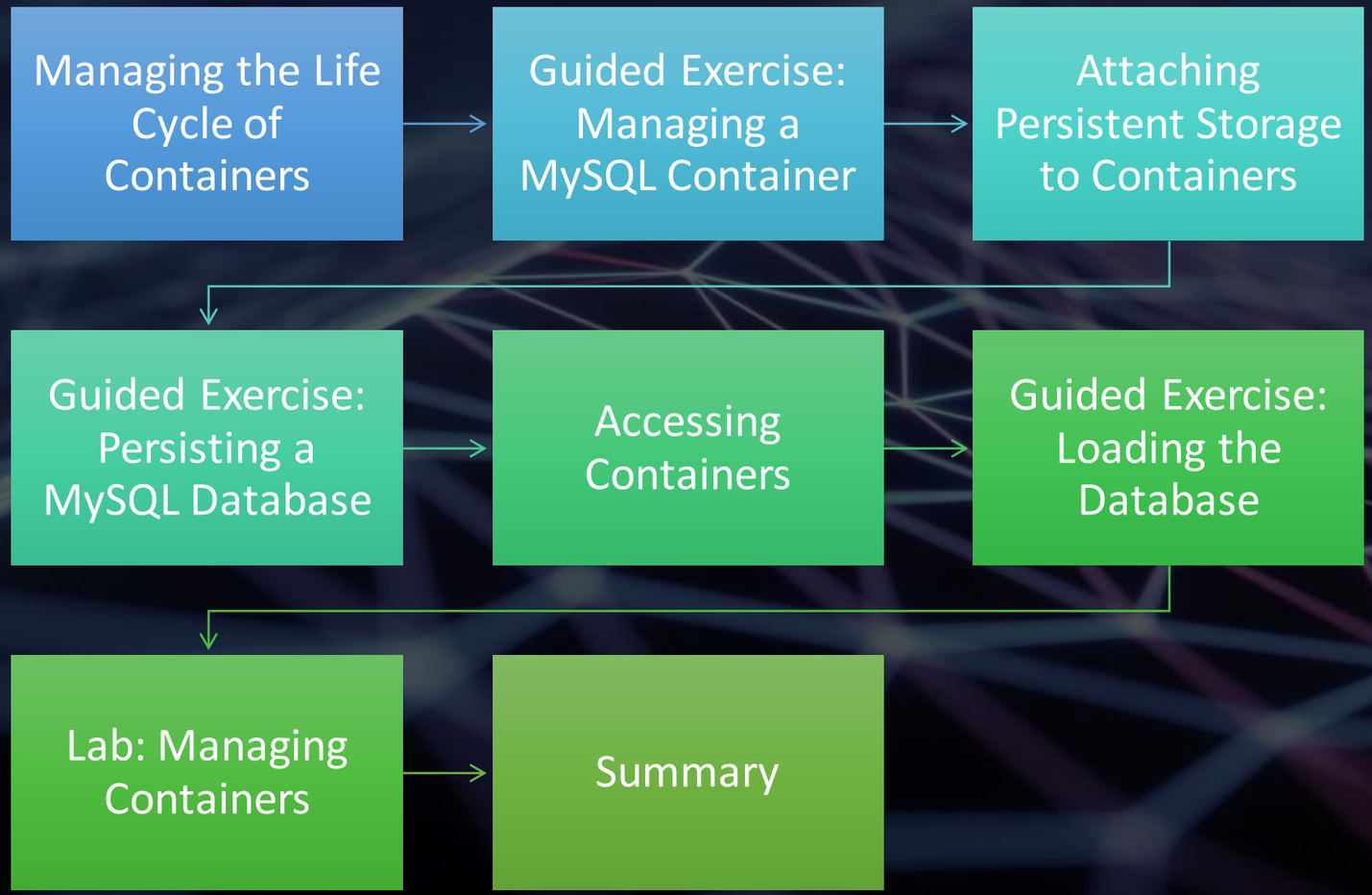
Managing Containers

Chapter objectives

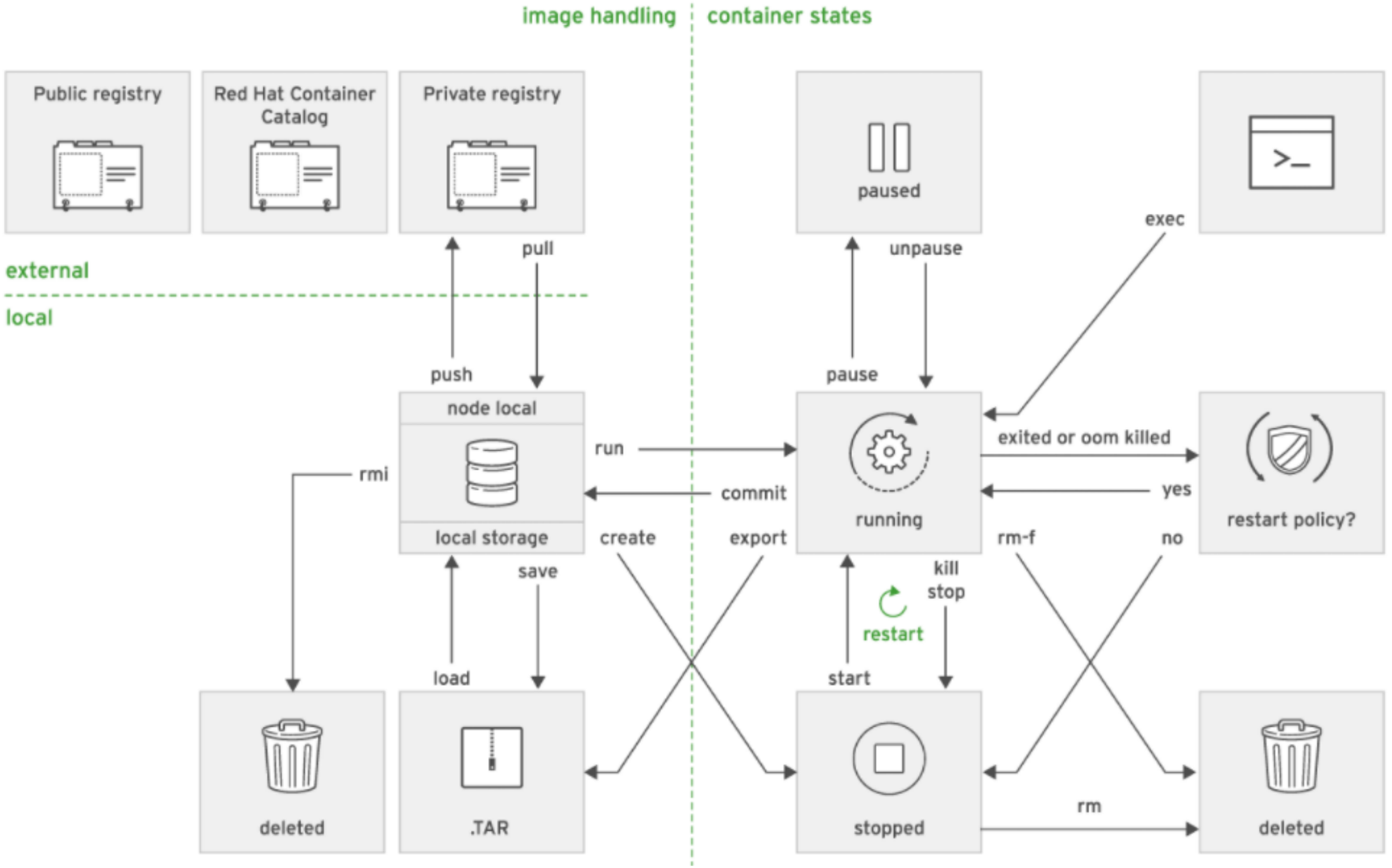


Managing the Life Cycle of Containers

After completing this section, you will be:

- Introduced to
  - Container Life Cycle Management with Podman
- Creating
  - Containers
- Running
  - Commands in a Container

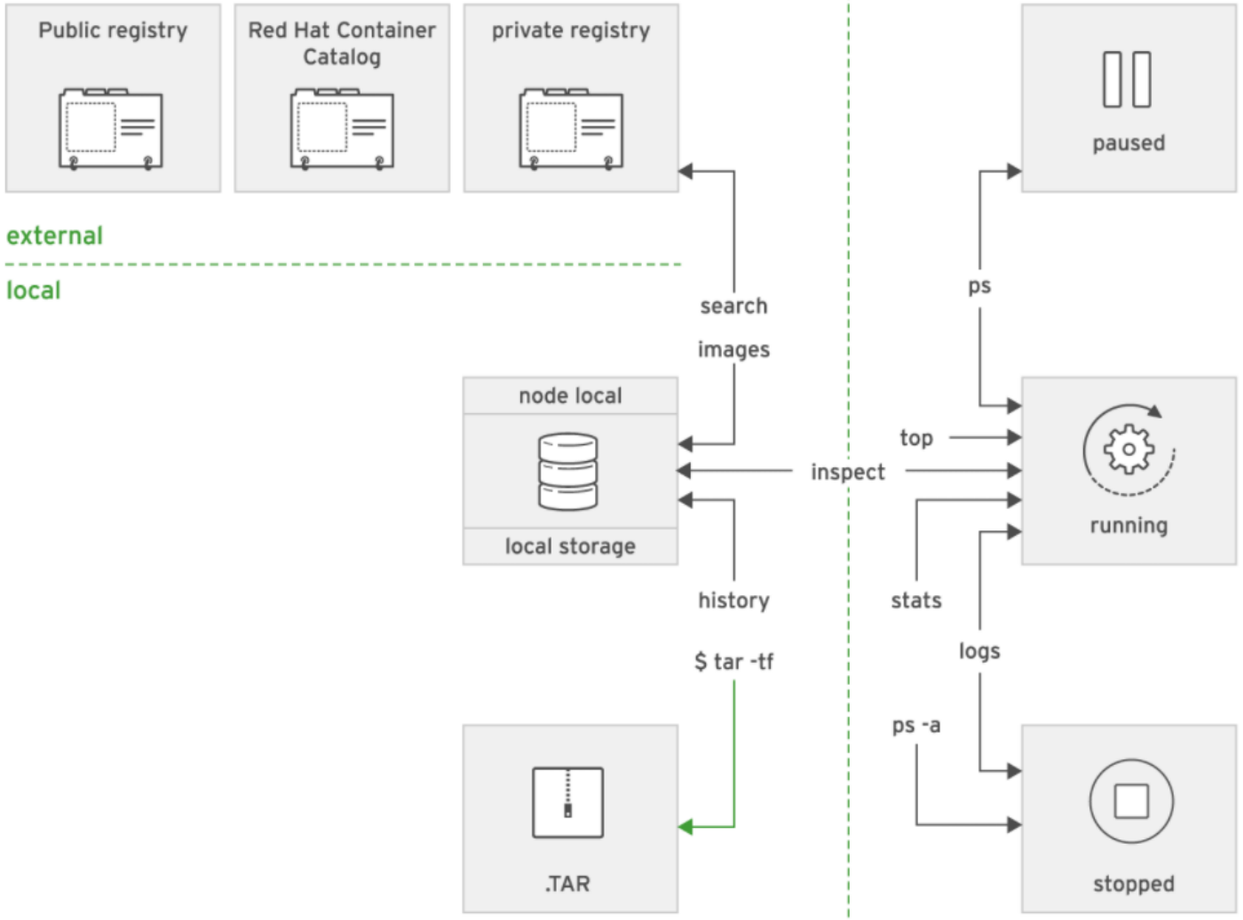
Container Life Cycle Management with Podman



Container Life Cycle Management with Podman

Subcommand	Description
run	Launch image into a container process. *
rm	Stop and removes running container. -f force removes it
stop	Stop running container
rmi	Removes local image.
build	Create a local image
pause	Suspend a running container. Process still in memory
unpause	Resume a suspended container
start	Restart a stopped container
save/load	Backup and restore local image . Create tar ball
export	Backup a running container into tar ball
push	Upload image into container registry
pull	Download image from container registry
kill	Gracefully shutdown processes in container. Clean shutdown
exec	Execute command or start a terminal session from running container

Gather information



Managing Containers - ps

List current running containers

```
[student@workstation ~]$ sudo podman ps
CONTAINER ID   IMAGE        COMMAND     CREATED  STATUS   PORTS    NAMES
77d4b7b8ed1f❶ rhscl/httpd-24-rhel7❷ "httpd..."❸ ...ago❹ Up...❺ 80/tcp❻ my-htt...❼
```

1. Each container, when created, gets a container ID, which is a hexadecimal number. This ID looks like an image ID but is unrelated.
2. Container image that was used to start the container.
3. Command executed when the container started.
4. Date and time the container was started.
5. Total container uptime, if still running, or time since terminated.
6. Ports that were exposed by the container or any port forwarding that might be configured.
7. The container name.

List all running and stopped containers

```
[student@workstation ~]$ sudo podman ps -a
```

Managing Containers - inspect

- Obtain configuration of container or image

```
[student@workstation ~]$ sudo podman inspect my-httpd-container
[
{
    "Id": "980e45...76c8be",
...output omitted...
    "NetworkSettings": {
        "Bridge": "",
        "EndpointID": "483fc9...5d801a",
        "Gateway": "172.17.42.1",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "HairpinMode": false,
        "IPAddress": "172.17.0.9",
...output omitted...
```

-

[student@workstation ~]$ sudo podman inspect “{{ .NetworkSettings.IPAddress }}” \

my-httpd-container

172.17.0.9

Managing Containers – kill or stop

- Send a unix KILL signal to processes in container

```
[ [student@workstation ~]$ sudo podman kill –s SIGKILL my-httpd-container
```

- Gracefully stop a running container

```
[ [student@workstation ~]$ sudo podman stop my-httpd-container
```

- -a : stop all running containers
- -f : force

Unix Kill signals

Signal	Value	Default Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at tty
SIGTTIN	21,21,26	Stop	tty input for background process
SIGTTOU	22,22,27	Stop	tty output for background process

Managing Containers – restart or start

- Restart a running container

```
[ [student@workstation ~]$ sudo podman restart --time 5 my-httpd-container
```

- Restart a stopped container

```
[ [student@workstation ~]$ sudo podman start my-httpd-container
```

- -a : start all running containers
- -f : force

Managing Containers – restart or start

- Remove a stopped container

```
[ [student@workstation ~]$ sudo podman rm my-httpd-container
```

- Remove a running container

```
[ [student@workstation ~]$ sudo podman rm –f my-httpd-container
```

- -a : stop all running containers
- -f : force

NOTE

The **inspect**, **stop**, **kill**, **restart**, and **rm** subcommands can use the container ID instead of the container name.

Guided Exercise: Managing a MySQL Container

You should be able to:

Start MySQL database container.

Troubleshoot and fix issue.

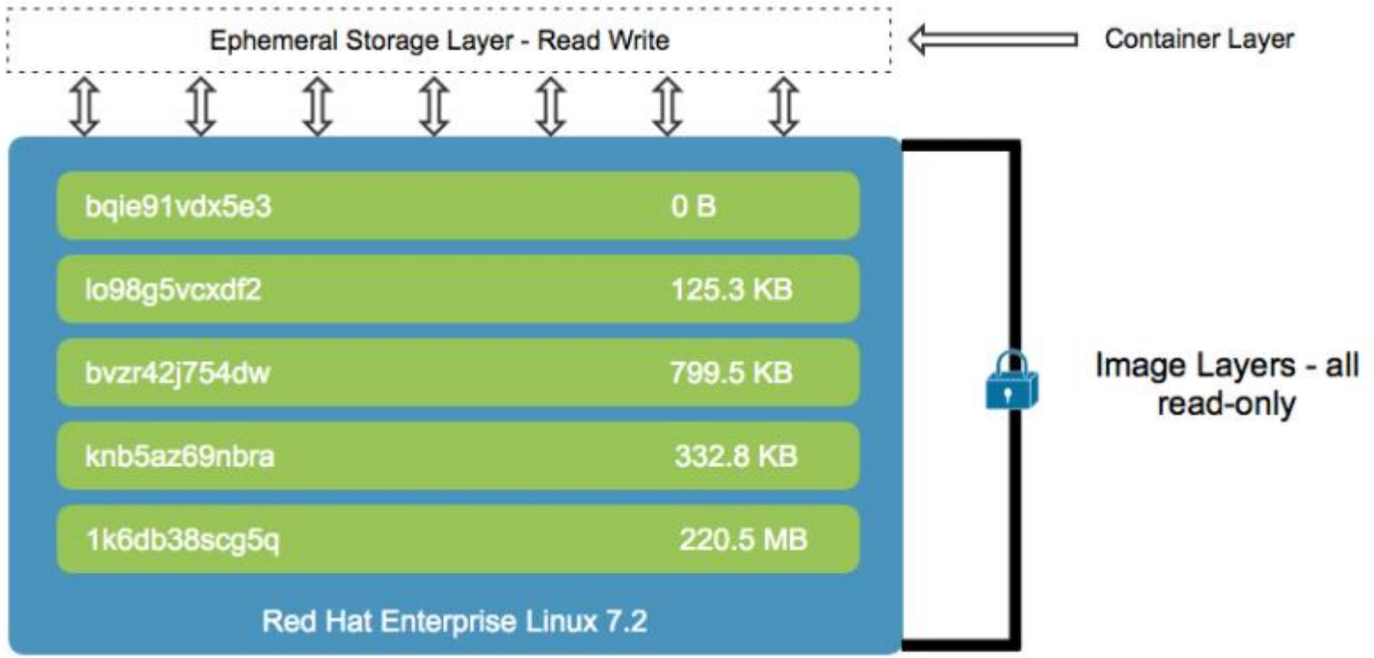
Populate database.

Attaching Persistent Storage to Containers

After completing this section, you will be:

- Save application
  - data across container restarts through the use of persistent storage.
- Configure
  - host directories for use as container volumes.
- Mount
  - a volume inside the container.

Persistent Storage / Volume



Container storage is ephemeral

- Do not use container storage to store persistent data
- Reclaiming storage

```
[student@workstation ~]$ sudo podman ps -a
```

Preparing the Host Directory

1. Create a directory with owner and group `root`:

```
[student@workstation ~]$ sudo mkdir /var/dbfiles
```

2. The user running processes in the container must be capable of writing files to the directory. If the host machine does not have exactly the same user defined, the permission should be defined with the numeric user ID (UID) from the container. In the case of the Red Hat-provided MySQL service, the UID is 27:

```
[student@workstation ~]$ sudo chown -R 27:27 /var/dbfiles
```

3. Apply the `container_file_t` context to the directory (and all subdirectories) to allow containers access to all of its contents.

```
[student@workstation ~]$ sudo semanage fcontext -a -t container_file_t '/var/dbfiles(/.*)?'
```

4. Apply the SELinux container policy that you set up in the first step to the newly created directory:

```
[student@workstation ~]$ sudo restorecon -Rv /var/dbfiles
```

Mounting a Volume

- Mount this directory to container

```
[student@workstation ~]$ sudo podman run -v "/var/dbfiles:/var/lib/mysql"  rhmap47/mysql
```

-v flag to mount volume

/var/dbfiles is the host directory

/var/lib/mysql is the mount point inside container

rhmap47/mysql is the image name

Guided Exercise: Persisting a MySQL Database

You should be able to:

- Prepare the Host directory
- Deploy container with a persistent database.

Accessing Containers

After completing this section, you will be:

Describe

- basics networking with containers.

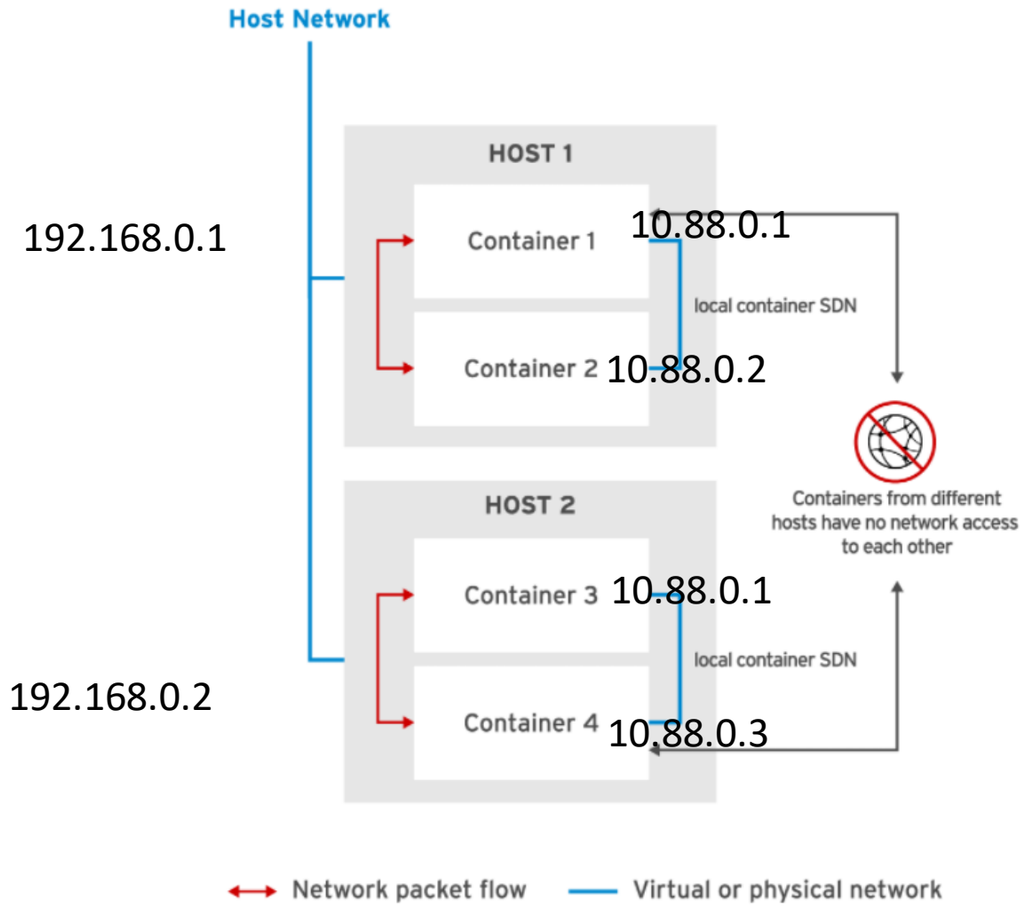
Remotely connect

- to services within a container.

Introducing Networking with Containers

- Container Networking Interface (CNI)
- Cloud Native Computing Foundation (CNCF)
  - sponsors CNI open source projects
- Standardized network interface
- Software Defined Networking (SDN)
- Podman command uses CNI
  - Settings in /etc/cni/net.d/87-podman-bridge.conflist

Basic Linux Container Networking



Mapping Network Ports

- Container have unique IP
- Each host configure unique isolated SDN
- SDN is only accessible from container host
- What is the solution?

Solution: Port forwarding

-p local_host_ip:host_port:container_target_port

```
[student@workstation ~]$ sudo podman run --name apache1 \
-p 8080:80   -d rhscl/httpd-24-rhel7:2.4

[student@workstation ~]$ sudo podman run --name apache2 \
-p 8181:80   -d rhscl/httpd-24-rhel7:2.4

[student@workstation ~]$ sudo podman run --name apache3 \
-p 127.0.0.1::80   -d rhscl/httpd-24-rhel7:2.4

[student@workstation ~]$ sudo podman run --name apache4 \
-p 80   -d rhscl/httpd-24-rhel7:2.4

[student@workstation ~]$ sudo podman port apache3

80/tcp  -> 127.0.0.1:35134
```

Guided Exercise: Loading the Database

You should be able to:

- Create a MySQL database container with port forwarding enabled.
- Populate database with a SQL script.

Lab: Managing Containers

You should be able to:

- Deploy and manage a persistent database using a shared volume.
- Start a second database using the same shared volume
- Observe that the data is consistent between the two containers because they are using the same directory

Chapter Summary

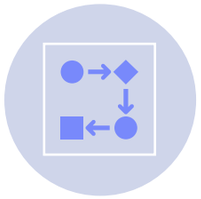
In this chapter, you learned:

Podman has subcommands to: create a new container (run), delete a container (rm), list containers (ps), stop a container (stop), and start a process in a container (exec).

Default container storage is ephemeral, meaning its contents are not present after the container restarts or is removed.

Containers can use a folder from the host file system to work with persistent data.

Podman mounts volumes in a container with the -v option in the podman run command.



The podman exec command starts an additional process inside a running container.

Podman maps local ports to container ports by using the -p option in the run subcommand.