Managing the Container Lifecycle

Objectives

- List, stop, and delete containers with Podman.

Container Lifecycle

Podman provides a set of subcommands to create and manage containers. You can use those subcommands to manage the container and container image lifecycle.

The following figures shows a summary of the most commonly used Podman subcommands that change the container and image state:

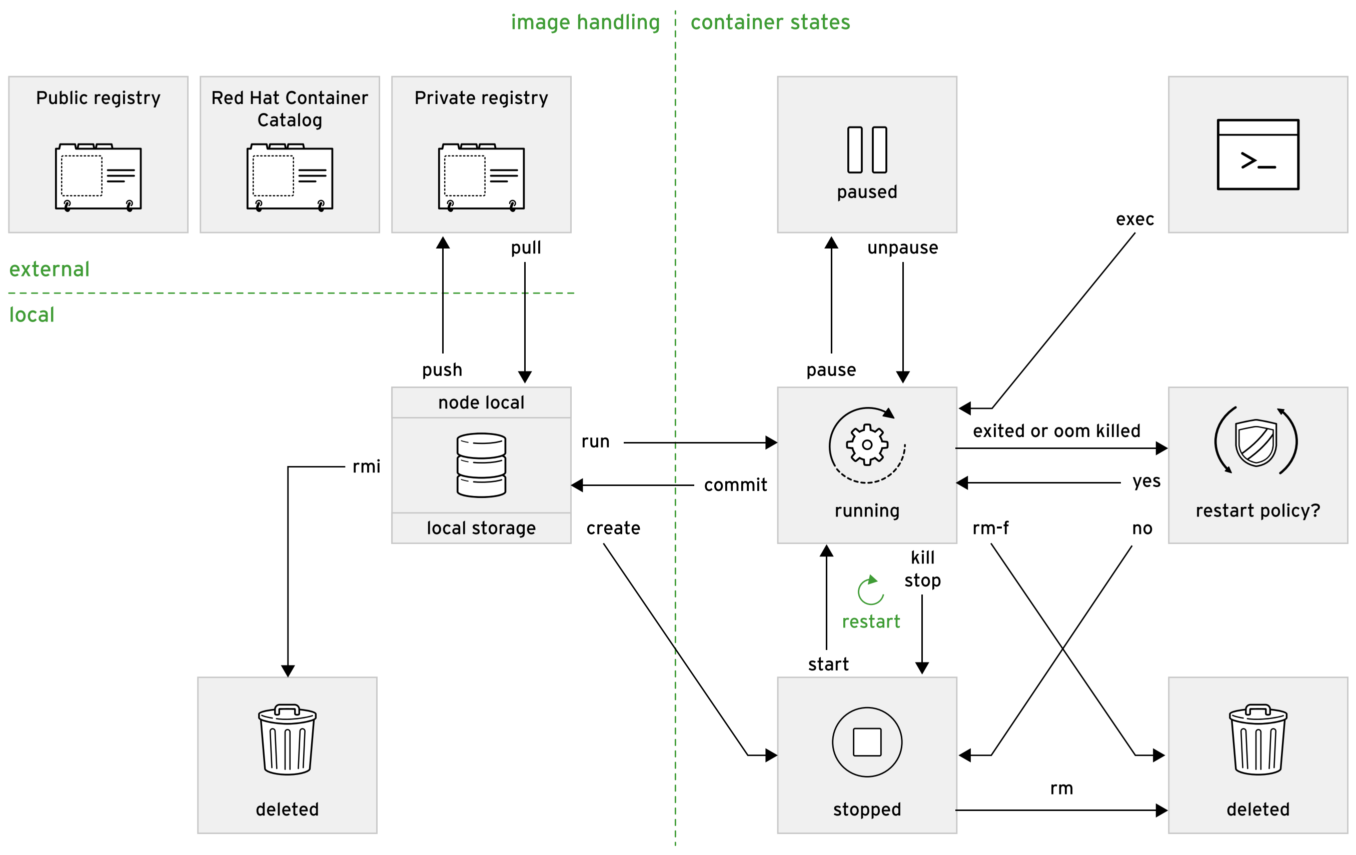

Figure 2.10: Podman lifecycle commands

Podman also provides a set of subcommands to obtain information about running and stopped containers.

You can use these subcommands to extract information from containers and images for debugging, updating, or reporting purposes. The following figure shows a summary of the most commonly used subcommands that query information from containers and images:

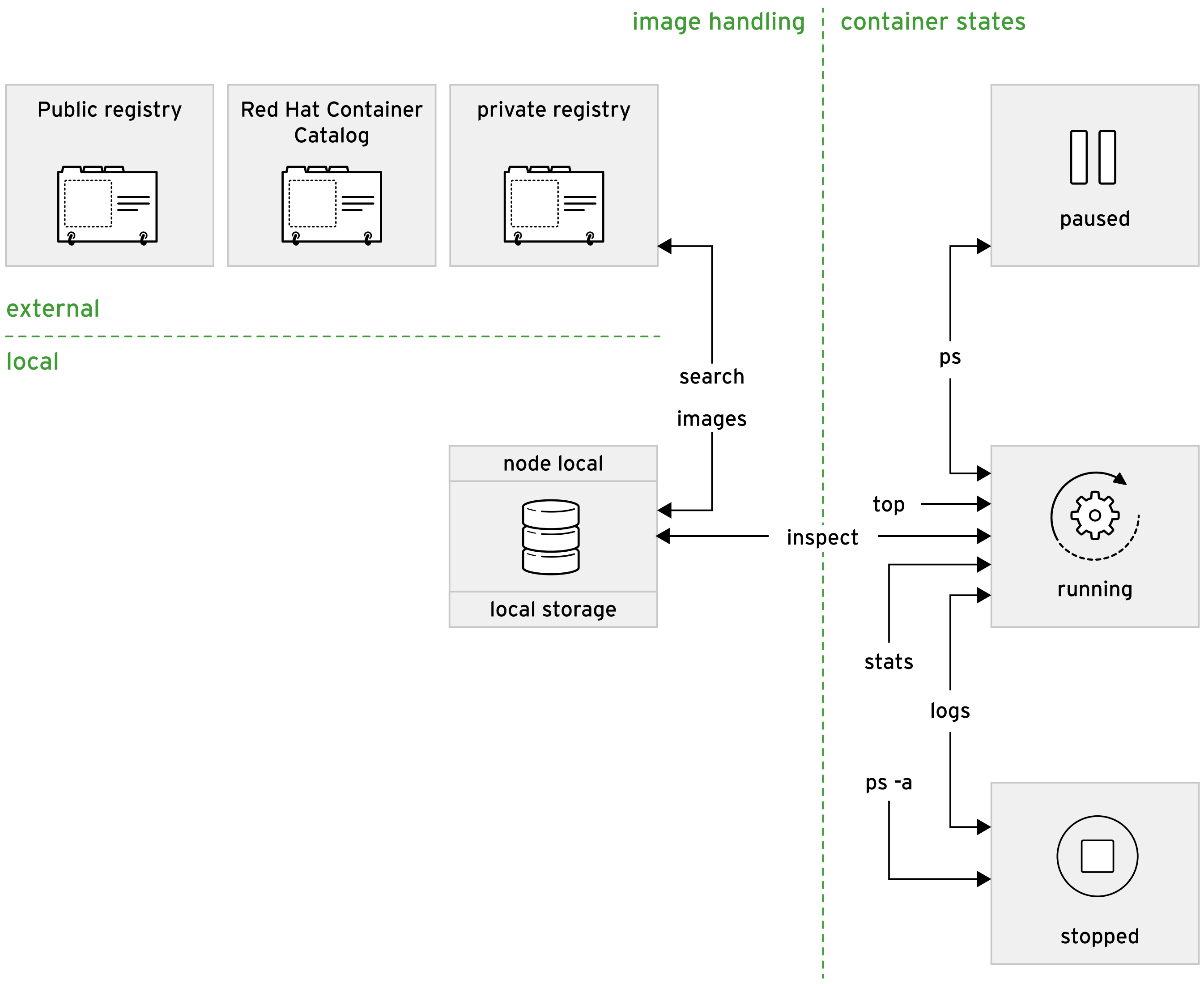

Figure 2.11: Podman query commands

This lecture covers the basic operations that you can use to manage containers. Commands explained in this lecture accept either a container ID or the container name.

Listing Containers

You can list running containers with the `podman ps` command.

```
[user@host ~]$ podman ps
CONTAINER ID  IMAGE         COMMAND              CREATED STATUS PORTS       NAMES
0ae7be593698  ...server     /bin/sh -c python... ...ago  Up...  ...8000/tcp httpd
c42e7dca12d9  ...helloworld /bin/sh -c nginx...  ...ago  Up...  ...8080/tcp nginx
```

Each row describes information about the container, such as the image used to start the container, the command executed when the container started, and the container uptime.

You can include stopped containers in the output by adding the `--all` or `-a` flag to the `podman ps` command.

```
[user@host ~]$ podman ps --all
CONTAINER ID  IMAGE         COMMAND              CREATED STATUS    PORTS        NAMES
0ae7be593698  ...server     /bin/sh -c python... ...ago  Up...     ...8000/tcp  httpd
bd5ada1b6321  ...httpd-24   /usr/bin/run-http... ...ago  Exited... ...8080/tcp  upbea
t...
c42e7dca12d9  ...helloworld /bin/sh -c nginx ... ...ago  Up...     ...8080/tcp  nginx
```

Inspecting Containers

In the context of Podman, inspecting a container means retrieving the full information of the container. The `podman inspect` command returns a JSON array with information about different aspects of the container, such as networking settings, CPU usage, environment variables, status, port mapping, or volumes. The following snippet is a sample output of the command.

```
[user@host ~]$ podman inspect 7763097d11ab
[
    {
        "Id": "7763...cbc0",
        "Created": "2022-05-04T10:00:32.988377257-03:00",
        "Path": "container-entrypoint",
        "Args": [
            "/usr/bin/run-httpd"
        ],
        "State": {
            "OciVersion": "1.0.2-dev",
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 9746,
...output omitted...
        "Image": "d2b9...fa0a",
        "ImageName": "registry.access.redhat.com/ubi8/httpd-24:latest",
        "Rootfs": "",
...output omitted...
            "Env": [
                "PATH=/opt/app-root/src/bin:/opt/app-root/bin:/usr/local/sbin:/usr/loca
l/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "TERM=xterm",
                "container=oci",
                "HTTPD_VERSION=2.4",
...output omitted...
            "CpuCount": 0,
            "CpuPercent": 0,
            "IOMaximumIOps": 0,
            "IOMaximumBandwidth": 0,
            "CgroupConf": null
        }
    }
]
```

The previous JSON output omits most fields. Because the full JSON is long, you might find it difficult to find the specific fields that you want to retrieve. You can pass a `--format` flag to the `podman inspect` command to filter the command output. The `--format` flag expects a Go template expression as a parameter, which you can use to select specific fields in the JSON.

Go template expressions use annotations, which are delimited by double curly braces (`{{` and `}}`), to refer to elements, such as fields or keys, in a data structure. The data structure elements are separated by a dot (`.`) and must start in upper case.

In the following command, the `Status` field of the `State` object is retrieved for a container called `redhat`.

```
[user@host ~]$ podman inspect \
 --format='{{.State.Status}}' redhat
running
```

In the preceding example, the `podman inspect` command uses the supplied Go template to navigate the JSON array of the `redhat` container. The template returns the `Status` field value of the `State` object from the JSON array.

Refer to the references section for more information about the Go templating language.

Stopping Containers Gracefully

You can stop a container gracefully by using the `podman stop` command. When you execute the `podman stop` command, Podman sends a `SIGTERM` signal to the container. Processes use the `SIGTERM` signal to implement cleanup procedures before stopping.

The following command stops a container with a container ID `1b982aeb75dd`.

```
[user@host ~]$ podman stop 1b982aeb75dd
1b982aeb75dd
```

You can stop all the running containers by using the `--all` or `-a` flag. In the following example, the command stops three containers.

```
[user@host ~]$ podman stop --all
4aea...0c2a
6b18...54ea
7763...cbc0
```

If a container does not respond to the `SIGTERM` signal, then Podman sends a `SIGKILL` signal to forcefully stop the container. Podman waits 10 seconds by default before sending the `SIGKILL` signal. You can change the default behavior by using the `--time` flag.

```
[user@host ~]$ podman stop --time=100
```

In the previous example, Podman gives the container a grace period of 100 seconds before sending the killing signal.

Stopping a container differs from removing a container. Stopped containers are present on the host machine and can be listed by using the `podman ps --all` command.

Additionally, when the containerized process finishes, the container enters the exited state. Such a process might finish for a number of reasons, such as an error, OOM (out-of-memory) state, or successfully finishing. Podman lists exited containers with other stopped containers.

Stopping Containers Forcefully

You can send the `SIGKILL` signal to the container by using the `podman kill` command. In the following example, a container called `httpd` is stopped forcefully.

```
[user@host ~]$ podman kill httpd
httpd
```

Pausing Containers

Both `podman stop` and `podman kill` commands eventually send a `SIGKILL` signal to the container. The `podman pause` command suspends all processes in the container by sending the `SIGSTOP` signal.

```
[user@host ~]$ podman pause 4f2038c05b8c
4f2038c05b8c
```

The `podman unpause` command resumes a paused container.

```
[user@host ~]$ podman unpause 4f2038c05b8c
4f2038c05b8c
```

Restarting Containers

Execute the `podman restart` command to restart a running container. You can also use the command to start stopped containers.

The following command restarts a container called `nginx`.

```
[user@host ~]$ podman restart nginx
1b98...75dd
```

Removing Containers

Use the `podman rm` command to remove a stopped container. The following command removes a stopped container with the container ID `c58cfd4b90df`.

```
[user@host ~]$ podman rm c58cfd4b90df
c58c...3150
```

You cannot remove running containers by default. You must stop the running container first and then remove it. The following command tries to remove a running container.

```
[user@host ~]$ podman rm c58cfd4b90df
Error: cannot remove container c58c...3150 as it is running - running or paused contain
ers cannot be removed without force: container state improper
```

You can add the `--force` (or `-f`) flag to remove the container forcefully.

```
[user@host ~]$ podman rm c58cfd4b90df --force
c58c...3150
```

You can also add the `--all` (or `-a`) flag to remove all stopped containers. This flag fails to remove running containers. The following command removes two containers.

```
[user@host ~]$ podman rm --all
6b18...54ea
6c0d...a6fb
```

You can combine the `--force` and `--all` flags to remove all containers, including running containers.

Quadlets

In a traditional environment, administrators configure applications such as web servers or databases to start at boot time, and run indefinitely as a systemd service. This is especially important in edge devices because containers need full lifecycle management, but a full orchestration solution might consume excessive computing resources.

Systemd is a system and service management tool for Linux operating systems. Systemd uses service unit files to start and stop applications, or to enable them to start at boot time. Typically, an administrator manages these applications with the `systemctl` command.

Quadlets bridge the integration between systemd and containers by enabling the creation of declarative configuration to manage containers as system applications.

Quadlet Unit File

A Quadlet unit file is a systemd unit file, but with a custom `Container` section to specify the container properties.

The following is an example Quadlet unit file for the httpd container image:

```
[Unit]
Description=One time container that outputs the contents of the OS release file

[Container]
Image=registry.access.redhat.com/ubi8/httpd-24:latest ❶
Volume=/www:/var/www/html:ro ❷
Entrypoint=/usr/bin/cat ❸
Exec=/etc/os-release ❹

[Install]
WantedBy=multi-user.target
```

❶ Indicates the container image to use as the service

❷ Maps the `/www` local path to the `/var/www/html` directory within the container in read-only mode

❸ Overrides the container's starting command, called the entrypoint

4 Adds additional parameters for the entrypoint

The Quadlet unit files use the `<serviceName>.container` naming pattern and can be stored in the following paths:

Rootful	Rootless
/run/containers/systemd/	$XDG_RUNTIME_DIR/containers/systemd/
/etc/containers/systemd/	$XDG_CONFIG_HOME/containers/systemd/
/usr/share/containers/systemd/	/etc/containers/systemd/users/$(UID)
	/etc/containers/systemd/users/

After adding or modifying unit files, you must use the `systemctl` command to reload the systemd configuration.

```
[user@host ~]$ systemctl --user daemon-reload
```

Managing the Quadlet Service

To manage a Quadlet service, use the `systemctl command`.

```
[user@host ~]$ systemctl --user [start, stop, status, enable, disable] container-
web.service
```

When you use the `--user` option, by default, systemd starts the service at your login, and stops it at your logout. You can start your enabled services at the operating system boot, and stop them on shutdown, by running the `loginctl enable-linger` command.

```
[user@host ~]$ loginctl enable-linger
```

To revert the operation, use the `loginctl disable-linger` command.

REFERENCES

podman-inspect(1) man page

podman-stop(1) man page

podman-restart(1) man page

podman-rm(1) man page

Go templates package

Systemd units using Podman Quadlet