How to run pods as systemd services with Podman

Extending traditional Linux system administration practices with the modern world of containers is a natural evolution.

Posted: March 16, 2022 | | Valentin Rothberg (Sudoer alumni, Red Hat)

Image by bluebudgie from Pixabay

Podman is well known for its seamless integration into modern Linux systems, and supporting systemd is a cornerstone in these efforts. Linux commonly uses the systemd init system to manage local services such as web servers, container engines, network daemons, and all of their interdependencies. Extending these more traditional Linux system administration practices with the modern world of containers is a natural evolution.

Linux containers

- A practical introduction to container terminology
- Containers primer
- Download now: Red Hat OpenShift trial
- eBook: Podman in Action
- Why choose Red Hat for containers

There are two common use cases for combining systemd and containers:

- Running systemd inside a container
- Using systemd to run containerized applications

The first scenario is running systemd inside of a container. As Dan Walsh explains, running systemd inside a container is as simple as it can be when using Podman. Podman automatically sets up several mounts in the container, and systemd is good to go. While it's a comparatively small Podman feature, it was a huge leap for running containerized workloads when it was introduced.

Historically, other container tools have not supported systemd. Users faced the challenge of writing custom init scripts, which are prone to errors and a support burden for software vendors. With Podman, all these issues go away. Users can use systemd to install and run their applications in containers, just like anywhere else, and software vendors will not face the challenges of dealing with custom init scripts written by their users.

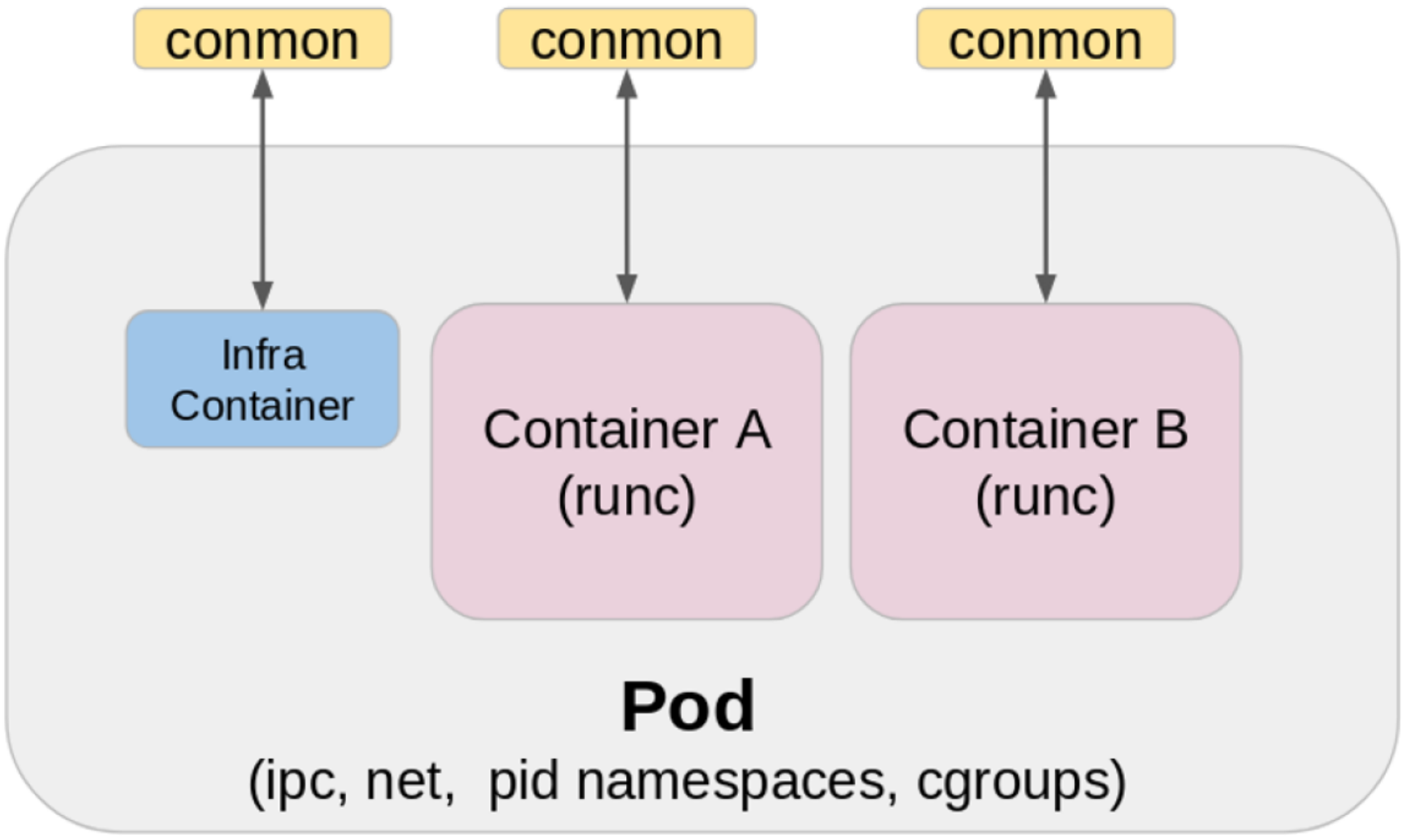
The second scenario is using systemd to run and manage containerized applications. That means systemd starts a containerized application and manages its entire lifecycle. Podman simplifies this with the `podman generate systemd` command, which generates a systemd unit file for a specified container or pod. Podman v1.7 and later support generating such units. Over time, our team has improved this feature and generated systemd unit files that can run on other machines, similar to using a Kubernetes YAML or a Compose file. Furthermore, the tight integration with systemd laid the foundation for auto-updates and simple rollbacks, supported since Podman v3.4.

While there are many earlier blogs and articles on generating systemd units for containers, there are none for generating these units for pods. But before going into these details, I want to review what a pod is.

[Getting started with containers? Check out this free course. Deploying containerized applications: A technical overview.]

What is a pod?

There are several different parts in a pod, and I think Brent Baude explains it best with the great figure below:



(Brent Baude, CC BY-SA 4.0)

The first thing to notice is that a pod consists of one or more containers. The group shares control groups (cgroups) and specific namespaces such as the PID, network, and IPC namespace. The shared cgroups ensure that all containers have the same resource constraints. The shared namespaces allow the containers to communicate with each other more easily (such as through localhost or interprocess communication).

You can also see a special infra container. Its primary purpose is to hold specific resources associated with the pod open, such as

ports, namespaces, or cgroups. The infra container is the pod's top-level container, and it's created before other containers and destroyed last. You use the infra container when generating systemd units for a pod, so keep in mind that this container runs for the pod's entire lifespan. It also implies that you cannot generate systemd units for pods without an infra container (such as `--infra=false`).

Last but not least, you see multiple `conmon` processes running, one per container. "Common" is short for container monitor, which sums up its main functionality. It also takes care of forwarding logs and performing cleanup actions once the container has exited. The `conmon` process starts before the container and instructs the underlying container runtime (such as `runc` or `crun`) to create and start the container. It also exits with the container's exit code allowing for using it as a systemd service's main process.

Generating systemd units for a pod

Podman generates exactly one system unit for a container. Once installed, use `systemctl` to start, stop, and inspect the service. The main PID of each unit is the container's conmon process. This way, systemd can read the container's exit code and act according to the configured restart policy. For more details on the units, please refer to Running containers with Podman and systemd shareable services and Improved systemd Podman with Podman 2.0.

Generating units for a pod is very similar to starting a container. Each container in the pod has a dedicated systemd unit, and each unit depends on the pod's main systemd unit. This way, you can continue using `systemctl` to start, stop, and inspect the pod's main service; systemd will take care of (re)starting and stopping the containers' services along with the main service.

This example creates a pod with two containers, generates unit files for the pod, and then installs the files for the current user:

```
$ podman pod create --name=my-pod
635bcc5bb5aa0a45af4c2f5a508ebd6a02b93e69324197a06d02a12873b6d1f7

$ podman create --pod=my-pod --name=container-a -t centos top
c04be9c4ac1c93473499571f3c2ad74deb3e0c14f4f00e89c7be3643368daf0e

$ podman create --pod=my-pod --name=container-b -t centos top
b42314b2deff99f5877e76058ac315b97cfb8dc40ed02f9b1b87f21a0cf2fbff

$ cd $HOME/.config/systemd/user

$ podman generate systemd --new --files --name my-pod
/home/vrothberg/.config/systemd/user/pod-my-pod.service
/home/vrothberg/.config/systemd/user/container-container-b.service
/home/vrothberg/.config/systemd/user/container-container-a.service
```

As expected, Podman generated three `.service` files, one for each container plus the top-level one for the pod. Please refer to the appendix at the end of the article to see the entire contents of the unit files. The units generated for the two containers look like standard container units plus the following systemd dependencies:

- `BindsTo=pod-my-pod.service`: The container unit is "bound" to the unit of the pod. If the pod's unit is stopped, this unit will be stopped, too.
- `After=pod-my-pod.service`: The container unit starts after the unit of the pod.

The pod's main service's dependencies further make sure that if a container unit does not start successfully, the main pod's main

unit will also fail.

That is all you need to know about generating systemd units for pods with Podman. Once you've reloaded systemd via `systemctl --user daemon-reload`, start and stop the `pod.service` at will. Have a look:

```
# Reload the daemon
$ systemctl --user daemon-reload

# Start the pod service and make sure the service is running
$ systemctl --user start pod-my-pod.service

$ systemctl --user is-active pod-my-pod.service
active

# Make sure the pod and its containers are running
$ podman pod ps
POD ID          NAME            STATUS          CREATED         INFRA ID        # OF CONTAINERS
6dd1090d4ca6  my-pod    Running         2 minutes ago  85f760a5cfe5  3
user $ podman container ps
CONTAINER ID  IMAGE                                   COMMAND         CREATED
STATUS          PORTS           NAMES
85f760a5cfe5  localhost/podman-pause:4.0.2-1646319369          5 minutes ago  Up 5 minutes
ago             6dd1090d4ca6-infra
44a7e60b9563  quay.io/centos/centos:latest            top             5 minutes ago  Up 5 minutes
ago             container-b
31f24bdff747  quay.io/centos/centos:latest            top             5 minutes ago  Up 5 minutes
ago             container-a
```

Great, everything is working as expected. You can use `systemctl` to start the services, and Podman lists the pods and their containers correctly. For the sake of consistency, have a final look at how to stop the pod service.

```
# Stop the pod service
$ systemctl --user stop pod-my-pod.service

# Make sure the pod and its containers are removed
$ podman pod ps -q

$ podman container ps -q

# Make sure the services are inactive
$ systemctl --user is-active pod-my-pod.service container-container-a.service container-container-
b.service
inactive
inactive
inactive
```

The take-home message is that Podman generates systemd units for pods just as it does for containers. The dependencies among these units are set in a way that you just need to interact with the pod's main unit, and systemd takes care of starting and stopping the containers' units.

Appendix

Podman generates the following unit files for a pod and the two related containers.

pod-my-pod.service

```
Description=Podman pod-my-pod.service
Documentation=man:podman-generate-systemd(1)
Wants=network-online.target
After=network-online.target
RequiresMountsFor=
Requires=container-container-a.service container-container-b.service
Before=container-container-a.service container-container-b.service

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStartPre=/bin/rm -f %t/pod-my-pod.pid %t/pod-my-pod.pod-id
ExecStartPre=/usr/bin/podman pod create --infra-conmon-pidfile %t/pod-my-pod.pid --pod-id-file
%t/pod-my-pod.pod-id --name=my-pod --replace
ExecStart=/usr/bin/podman pod start --pod-id-file %t/pod-my-pod.pod-id
ExecStop=/usr/bin/podman pod stop --ignore --pod-id-file %t/pod-my-pod.pod-id -t 10
ExecStopPost=/usr/bin/podman pod rm --ignore -f --pod-id-file %t/pod-my-pod.pod-id
PIDFile=%t/pod-my-pod.pid
Type=forking

[Install]
WantedBy=default.target
```

container-container-a.service

```
[Unit]
Description=Podman container-container-a.service
Documentation=man:podman-generate-systemd(1)
Wants=network-online.target
After=network-online.target
RequiresMountsFor=%t/containers
BindsTo=pod-my-pod.service
After=pod-my-pod.service

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStartPre=/bin/rm -f %t/%n.ctr-id
ExecStart=/usr/bin/podman run --cidfile=%t/%n.ctr-id --cgroups=no-conmon --rm --pod-id-file %t/pod-
my-pod.pod-id --sdnotify=conmon -d --replace --name=container-a -t centos top
ExecStop=/usr/bin/podman stop --ignore --cidfile=%t/%n.ctr-id
ExecStopPost=/usr/bin/podman rm -f --ignore --cidfile=%t/%n.ctr-id
Type=notify
NotifyAccess=all

[Install]
WantedBy=default.target
```

container-container-b.service

```
[Unit]
Description=Podman container-container-b.service
Documentation=man:podman-generate-systemd(1)
Wants=network-online.target
After=network-online.target
RequiresMountsFor=%t/containers
BindsTo=pod-my-pod.service
After=pod-my-pod.service

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStartPre=/bin/rm -f %t/%n.ctr-id
ExecStart=/usr/bin/podman run --cidfile=%t/%n.ctr-id --cgroups=no-conmon --rm --pod-id-file %t/pod-
my-pod.pod-id --sdnotify=conmon -d --replace --name=container-b -t centos top
ExecStop=/usr/bin/podman stop --ignore --cidfile=%t/%n.ctr-id
ExecStopPost=/usr/bin/podman rm -f --ignore --cidfile=%t/%n.ctr-id
Type=notify
NotifyAccess=all

[Install]
WantedBy=default.target
```

How Podman can transfer container images without a registry

The new 'podman image scp' command makes it easier to transfer container images between users on the same system or machines over the network.

Posted: March 11, 2022
Authors: Charlie Doern (Red Hat), Dan Walsh (Red Hat)

5 Podman features to try now

Improve how you use containers with these new Podman features: --latest, --replace, --all, --ignore, and --tz.

Posted: February 15, 2022
Author: Dan Walsh (Red Hat)

Podman 4.0's new network stack: What you need to know

Podman's new Netavark and Aardvark-based stack offers three main advantages over the existing CNI-based stack.

Posted: March 17, 2022
Author: Matthew Heon (Red Hat)

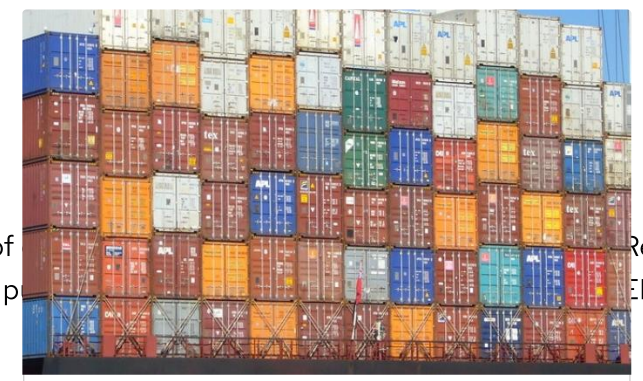
Topics: Podman Containers Linux

Valentin Rothberg

Container engineer at Red Hat, bass player, music lover. More about me

Related Content

e of
nal p



Red H
ENDE

Deploy an application in Red Hat OpenShift on your laptop

Now that your environment has been set up, deploy a sample application on an OpenShift Local cluster.

Posted: April 25, 2023
Author: Ricardo Gerardi (Editorial Team, Sudoer alumni, Red Hat)

f Red

How to install Red Hat OpenShift Local on your laptop

Install Red Hat OpenShift Local on your own machine to test your work before deployment.

Posted: April 24, 2023
Author: Ricardo Gerardi (Editorial Team, Sudoer alumni, Red Hat)

her c

8 open source 'Easter eggs' to have fun with your Linux terminal

Hunt these 8 hidden or surprising features to make your Linux experience more entertaining.

Posted: April 10, 2023
Author: Ricardo Gerardi (Editorial Team, Sudoer alumni, Red Hat)

Red Hat Summit