Rootless Podman

Objectives

- Run rootless containers with Podman.

Container Workload Isolation

Before the rise in container technology popularity, a single application might be separated into tiers, such as front end, API or back end, and database tiers. Developers often deployed each of these application parts to virtualized machines that were dedicated to a single application.

Virtualized machines then ran an operating system with a kernel separate from the host operating system. This meant that one physical machine could host several virtual machines but each of the virtual machines contained a kernel and typically only a single application.

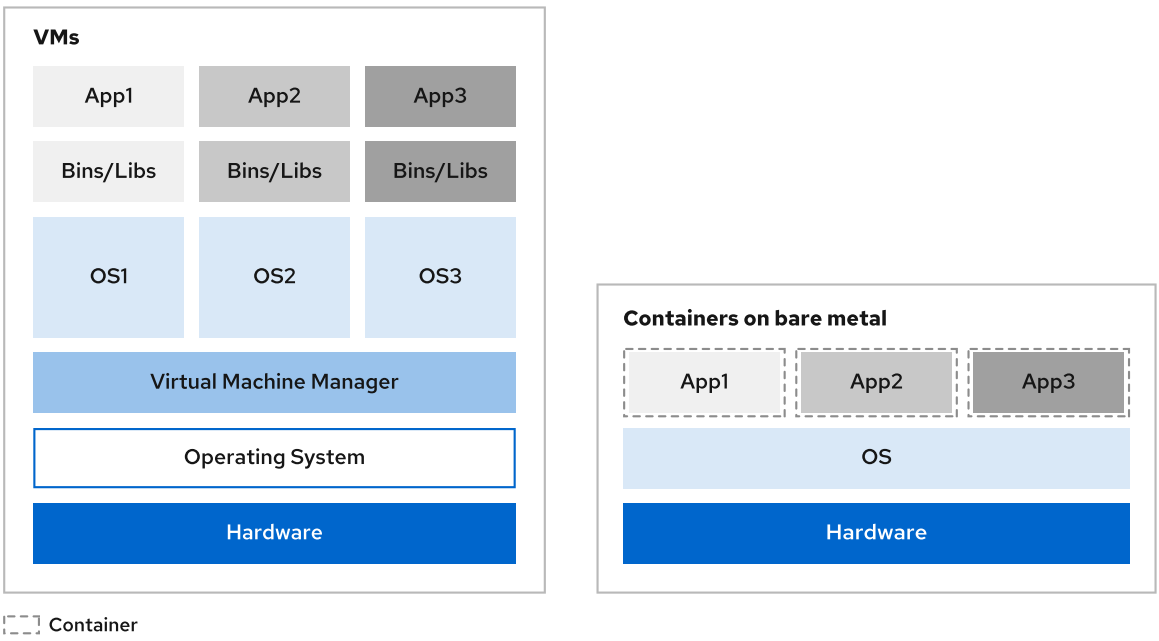

Figure 4.1: Container and VM application isolation

Container technology encourages developers to split a single application tier, such as back end, into multiple smaller microservices. Additionally, containerized processes use the kernel of the host operating system. Typically, due to the smaller resource requirements of a single containerized process, one host can run more containerized processes than it can virtual machines.

Because one machine can host multiple applications, exploiting a single application to gain superuser privilege on the host machine can lead to large-scale outages. Consequently, developers and administrators strive to create and maintain applications that use the principle of least privilege, which minimizes potential attack vectors.

Analyzing Rootless Containers

Rootless containers, or unprivileged containers, are containers that do not require administrator privileges.

A container is rootless only when it meets the following conditions:

- The containerized process does not use the root user, which is a special privileged user in Linux and UNIX systems. Such a user is for administrative purposes and has the ID *0*.
- The root user inside of the container is not the root user outside of the container.
- The container runtime does not use the root user. For example, if your container runtime runs as the root user, then containers managed by such runtime are not rootless containers.

Because Podman starts each container as a new process, the runtime does not require elevated privileges.

> **NOTE**
>
> The Podman process exits after creating a container. Then the container process ID attaches to the systemd parent process ID.

The following subsections examine the remaining requirements of rootless containers.

Prerequisites for Rootless Containers

Depending on your operating system, Podman might require host operating system setup to run rootless containers. The following list briefly describes the most common prerequisite setup.

cgroup v2
Cgroup v2 is a Linux kernel feature that Podman uses to limit container resource use without requiring elevating privileges.

Podman on Red Hat Enterprise Linux 9 (RHEL 9) uses cgroup v2 by default, with the `crun` container runtime implementation.

pasta
Podman uses the `pasta` command from the `passt` package to implement rootless networking for unprivileged network namespaces.

fuse-overlayfs
Podman uses the `fuse-overlayfs` package to manage the Copy-On-Write (COW) file system. Though Podman does not require this package, Red Hat recommends using it for performance reasons. COW file system is discussed later in the course.

See the references section for more information about rootless Podman setup.

Changing the Container User

When you create a Containerfile, the user tends to be root. This is because you require elevated privileges for certain operations, such as installing packages or making configuration changes.

Determine the current user by running the `id` command:

```
[user@host ~]$ podman run registry.access.redhat.com/ubi9/ubi id
uid=0(root) gid=0(root) groups=0(root)
```

The following container image uses the root user to start an HTTP server:

```
FROM registry.access.redhat.com/ubi9/ubi

CMD ["python3", "-m", "http.server"]
```

This is a security risk, because an attacker could exploit the application, get access to the container, and exploit further vulnerabilities to escape from the containerized environment into the host system. Attackers might escape the containerized environment by exploiting bugs and vulnerabilities typically in the kernel or container runtime, such as `crun`.

Consider the following Containerfile change:

```
FROM registry.access.redhat.com/ubi9/ubi

RUN adduser \
   --no-create-home \
   --system \
   --shell /usr/sbin/nologin \
   python-server

USER python-server

CMD ["python3", "-m", "http.server"]
```

Such a container starts the same process without elevated privileges, which makes it harder for an attacker to exploit a vulnerability and access the host system. The `RUN` instruction runs as the root user, because it precedes the `USER` instruction.

Explaining User Mapping

Traditionally, when an attacker gains access to the container file system by using an exploit, the root user inside the container corresponds to the root user outside of the container. This means that if an attacker escapes the container isolation, then they have elevated privileges on the host system, thus potentially causing more damage.

Podman maps users inside of the container to unprivileged users on the host system by using *subordinate ID* ranges. Podman defines the allowed ID ranges in the `/etc/subuid` and `/etc/subgid` files.

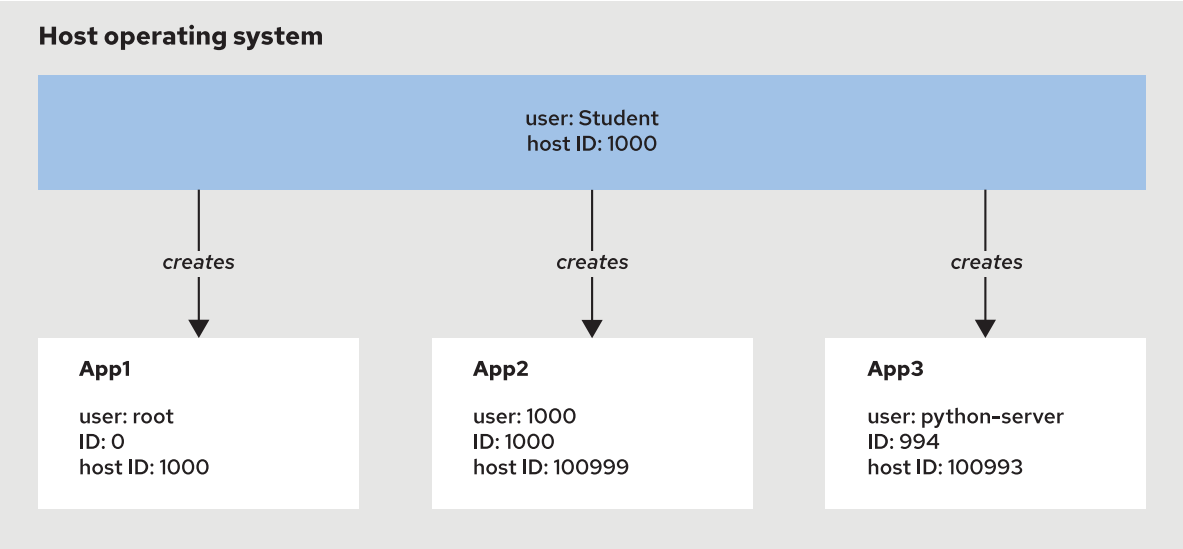

Figure 4.2: Examples of users with different IDs inside and outside of containers

Consider the following example:

```
[user@host ~]$ cat /etc/subuid /etc/subgid
student:100000:65536
student:100000:65536
```

On the system from the preceding example, the `student` user can allocate 65536 user IDs starting with the ID 100000. This leads to the `HostUserID = 100000 + ContainerUserID - 1` ID mapping. Consequently, user ID `1` in a container maps to the host user ID `100000`, and so on.

The user ID `0` (`root`) is an exception, because the `root` user maps to the user ID that started the container. For example, if a user with the ID `1000` starts a container that uses the `root` user, then the `root` user maps to the host user ID `1000`.

To generate the subordinate ID ranges, use the `usermod` command.

```
[user@host ~]$ sudo usermod --add-subuids 100000-165535 \
  --add-subgids 100000-165535 student
[user@host ~]$ grep student /etc/subuid /etc/subgid
/etc/subuid:student:100000:65536
/etc/subgid:student:100000:65536
```

The `/etc/subuid` and `/etc/subgid` files must exist before you define the subordinate ID ranges with the `usermod` command. After you define the ranges, you must execute the `podman system migrate` for the new subordinate ID ranges to take effect.

You can verify the mapped user by using the `podman top` command. For example, the following command starts a container and uses the `id` command to verify that the user inside of the container is `root`.

```
[user@host ~]$ podman run -it registry.access.redhat.com/ubi9/ubi bash
[root@e6116477c5c9 /]# id
uid=0(root) gid=0(root) groups=0(root)
```

Then, you must verify the mapping of host user (`huser`) to the container user (`user`). The following example uses the container ID `e6116477c5c9`:

```
[user@host ~]$ podman top e6116477c5c9 huser user
HUSER       USER
1000        root
```

The preceding example shows that the user inside the container, `root`, is mapped to a user with ID `1000` on the host system.

Alternatively, you can verify the same ID mapping by printing the `/proc/self/uid_map` and `/proc/self/gid_map` files inside of the container:

```
[root@e6116477c5c9 /]# cat /proc/self/uid_map /proc/self/gid_map
         0       1000          1
         0       1000          1
```

WARNING

When you execute a container with elevated privileges on the host machine, the root mapping does not take place even when you define subordinate ID ranges, for example:

```
[user@host ~]$ sudo podman run -it
registry.access.redhat.com/ubi9/ubi bash
[root@4746207beab7 /]# id
uid=0(root) gid=0(root) groups=0(root)
```

In a new terminal, verify the `podman top` output:

```
[user@host ~]$ sudo podman top 4746207beab7 huser user
HUSER       USER
root        root
```

Limitations of Rootless Containers

Rootless containers have limitations that make some applications unsuitable to be containerized as rootless containers. The following list describes some limitations of rootless containers.

Non-trivial Containerization

Some applications might require the root user. Depending on the application architecture, some applications might not be suitable for rootless containers or might require a deeper understanding to containerize.

For example, applications such as HTTPd and Nginx start a bootstrap process and then spawn a new process with a non-privileged user, which interacts with external users. Such applications are non-trivial to containerize for rootless use.

Red Hat provides containerized versions of HTTPd and Nginx that do not require root privileges for production usage. You can find the containers in the Red Hat container registry.

Required Use of Privileged Ports or Utilities

Rootless containers cannot bind to privileged ports, such as ports `80` or `443`. Red Hat recommends that you do not use privileged ports, and use port forwarding instead. However, if you require the use of privileged ports, then you can configure the unprivileged port range:

```
[user@host ~]$ sudo sysctl -w "net.ipv4.ip_unprivileged_port_start=79"
```

This means rootless containers can bind to port `80` and higher.

Similarly, rootless containers cannot use utilities that require the root user, such as the `ping` utility. This is because the `ping` utility requires elevated privileges to establish raw sockets, which is an action that requires the `cap_net_raw` privilege.

To solve such a requirement, verify whether you can grant the privilege to a non-root user. For example, you can specify a range of group IDs that are allowed to use the `ping` utility by using the `net.ipv4.ping_group_range` kernel parameter:

```
[user@host ~]$ sudo sysctl -w "net.ipv4.ping_group_range=0 2000000"
```

REFERENCES

Understanding root inside and outside a container

Application Container Security Guide

Podman Setup

Podman Troubleshooting

Shortcomings of Rootless Podman

Container Security Workshop