University Göttingen
GWDG AG-C
Freja Nordsiek

Exercise 1 / 2023.02.21
HPC System Administration – Containers / WiSe 2022/23
30 Minutes Total

Exercise Introduction

This is the exercise on using Podman (`https://podman.io`) and other tools in the Docker/OCI ecosystem. Tasks 1 and 2 can be done in either order, but should be done before Task 3. Task 4 should be done absolutely last since it depends on all of the others.

Contents

Task 1: Listing/deleting containers and container images (5 min)

It is very important to know what containers and container images one already has in local storage. Additionally, they can take up a lot of space (anywhere from 1 MiB to > 10 GiB) and one can delete ones that are no longer used. Podman provides the `KIND list` commands for this, where `KIND` is either `container` or `image` (for container images). Run these commands. For the container images, you should get something like

```
REPOSITORY                  TAG         IMAGE ID      CREATED      SIZE
docker.io/library/busybox  musl        611b54fbba47  6 weeks ago  1.63 MB
```

and for the containers, you should get something like

```
CONTAINER ID  IMAGE       COMMAND     CREATED     STATUS      PORTS       NAMES
```

Where did all the containers you made in the tutorial go? By default, Podman does not list container images for incompatible CPU architectures and OSes, and containers that are no longer alive. To list these, you must add the `-a` option (all). You will likely see the same thing for the container images, but for the containers you should finally see all the containers you used in the tutorial:

```
CONTAINER ID  IMAGE                           COMMAND      CREATED         STATUS                      PORTS       NAMES
6ba95297c325  docker.io/library/busybox:musl  ls -lh /bin  2 minutes ago  Exited (0) 2 minutes ago              ecstatic_pascal
a3b07090d0fe  docker.io/library/busybox:musl  /bin/sh      2 minutes ago  Exited (0) 2 minutes ago              gifted_shamir
8a2eeac8ea5c  docker.io/library/busybox:musl  sh           2 minutes ago  Exited (0) 2 minutes ago              NAME
```

In the tutorial, you were exclusively referring to container images by `REGISTRY/NAMESPACE:TAG` and containers by `NAME`. You can also refer to them using their `IMAGE ID` and `CONTAINER ID`, which will not change if you change the name and/or tag.

To delete containers and container images, Podman provieds the `KIND rm` commands. Delete one of the containers and container images.

Now, if you want to delete all containers or container images, you would add the `-a` option (all). Delete the rest of the dead containers.

If you really really really need to delete and reset everything, Podman also provides a `system reset` command.

Further Reading

- `podman container list` – `https://docs.podman.io/en/latest/markdown/podman-ps.1.html`
- `podman container rm` – `https://docs.podman.io/en/latest/markdown/podman-rm.1.html`
- `podman image list` – `https://docs.podman.io/en/latest/markdown/podman-images.1.html`
- `podman image rm` – `https://docs.podman.io/en/latest/markdown/podman-rmi.1.html`
- `podman system reset` – `https://docs.podman.io/en/latest/markdown/podman-system-reset.1.html`

Task 2: Evaluating base images on a registry (5 min)

You will look at and evaluate base images at docker hub. Docker hub's registry URL is the familiar `docker.io`. Its web URL is `https://hub.docker.com`.

Looking for numpy

Use its web URL and search for numpy container images (search box is at the top), and then look at the list you see. Scroll to the bottom, looking at the namespaces and names and then seeing how many there are.

Do any of these look like they might be official or from an otherwise trustwrothy source? What about the rest?

You may have noticed two near the top whose namespace is numpy, claiming to be from numpy. Take a minute to think about how you would go about verifying if they really are from numpy and they aren't just from a malware author who grabbed the name first or is using homoglyphs (e.g. the lower-case Greek beta and German Eszett).

Looking closer at alpine

Now, search for alpine (a small musl based linux distro). The first several have stamps. What do the stamps of the first (the one used in the presentation) and second imply? Does it seem trustworthy?

Look at the first one. Notice the black box at the top right that says `docker pull alpine`. That would be the Docker command (replace `docker` with `podman` for the Podman equivalent) to pull it, assuming you have the tool configured to use `docker.io` as the default repository when the repository is not specified. You will also notice that the namespace is missing (the default if not specified, is `library`). This is actually a very dangerous practice. Never do this. Always add in the registry URL (`docker.io` for docker hub) and namespace explicitly if not listed.

Try to find the `3.17` tag.

Further Reading

- docker hub – `https://hub.docker.com`
- alpine Linux – `https://alpinelinux.org`

Task 3: Build and run an nginx webserver in Podman (10 min)

You made trivial containers and container images in the tutorial. Now, it is time to actually make long running containers doing some task on a server. In this exercise, you will make a container running a simple webserver (nginx). The goal is to get it to run well enough to get a 404 message. Getting an actual webpage and other advanced things is not a goal of this exercise (you will learn more about a different webserver, Apache2, in the LAMP course later this week).

Making the container image with nginx

Make the `Dockerfile/Containerfile` for the container image. You should use the previously used `docker.io/library/alpine:3.17` since it is small and minimalistic. This alpine container image does not have nginx installed by default, so you will need to install it in the container using alpine's package manager `apt`. It works like `$ apt add PACKAGE`. Do not worry about configuring nginx yet.

There are two additional things you need to do that was not covered previously with `Dockerfile/Containerfiles`. One, you need to tell the container tool that the port for the webserver is to be exposed to the outside using the `EXPOSE` directive (port 80). Two, while you could specify the exact command every time you run the container, it is convenient to make the default command (when you specify nothing) be the command you want to run, which is nginx. Use the `CMD` directive to run nginx with no arguments. Its executable is located at `/usr/sbin/nginx` inside the container image. Don't forget to give it a name and tag when building it using the `--tag NAME:TAG` option.

Further Reading

- alpine Linux package database – `https://pkgs.alpinelinux.org`
- Dockerfile reference – `https://docs.docker.com/engine/reference/builder/`
- Dockerfile `CMD` – `https://docs.docker.com/engine/reference/builder/#cmd`
- Dockerfile `FROM` – `https://docs.docker.com/engine/reference/builder/#from`
- Dockerfile `EXPOSE` – `https://docs.docker.com/engine/reference/builder/#expose`
- Dockerfile `RUN` – `https://docs.docker.com/engine/reference/builder/#run`

Trial runs of failure

Now, run a container from your container image that you made with the `container run` command, but with the `-d` option (don't use `-i`, `-t`, or `--rm` options) so it starts in the detached state (it goes into the background once it is started). Then, list your containers. You will notice that it is dead (only shows up with the `-a` option).

The problem is that `/usr/sbin/nginx` starts the webserver in a separate background process and returns (the server daemonizes), which ends execution. Once the program that `podman container run` calls is done, the container is considered dead and all child processes are killed. To actually make the container survive, we have to make it so that /usr/sbin/nginx does not daemonize. Luckily, this is simple to set in nginx's config file /etc/nginx/nginx.conf. You just have to add the line daemon off; to it at the end. Change your Dockerfile/Containerfile to do just that and rebuild your container image.

Now, run the container the same way as before. You need to add the -p OUTSIDEPORT:INSIDEPORT option for 80:80 to export port 80 inside the container to outside the container (this is called publishing the port). This will either crash right away, or if it doesn't and you list your containers, you will see that it is there and running, like the following:

```
CONTAINER ID  IMAGE                   COMMAND          CREATED        STATUS            PORTS       NAMES
0a04bb915f7c  localhost/nginx:latest  /usr/sbin/nginx  4 seconds ago  Up 4 seconds ago              ng
```

which means it is running but pointing your web browser at your VM's IP address like http://IPADDRESS:PORT gets no response.

It failed because you ran the container as an uprivileged user (Podman is rootless afterall), and unprivileged users cannot bind to ports below 1024 (such as 80). The Linux permissions system is blocking your container, as it is designed. Stop and delete your container. Containers can be stopped using Podman's container stop command.

Further Reading

- podman container run – https://docs.podman.io/en/latest/markdown/podman-run.1.html
- podman container stop – https://docs.podman.io/en/latest/markdown/podman-stop.1.html

Remap to an unprivileged port

Podman and Docker can both map ports inside containers to different ports outside of them. This is in fact one of the cool things that containers allow one to do (e.g. have thousands of webservers running from the same container image but on different ports without having to make a unique container image for each one specifying the port). This remapping is done using the -p OUTSIDEPORT:INSIDEPORT option. Your VMs are setup so that only ports 22, 80, 443, and 8000-8010 are accessible from the outside world. Rerun the container using one of these other ports (**WARNING:** while the kernel should not let the container bind to port 22, don't actually try it in case there is a bug since then that would kill your login access to the VM). Then, point your browser at this new port and you should get a nice "404 Not Found" message from nginx (no pages were configured). When you have it working, stop and delete your container.

Further Reading

- podman container run – https://docs.podman.io/en/latest/markdown/podman-run.1.html

Task 4: Importing into Docker and running on port 80 (10 min)

Now, there are a lot of reasons you might want to actually use the privileged ports 1–1024 with your containers. There are a couple ways you could do this:

1. Make a firewall rule (nftables/iptables) that redirects incoming packets on port 80 to your unprivileged port.
2. Setup a reverse proxy to redirect requests on port 80 to the container's unprivileged port (often done with nginx on the host system).

3. Run the container in a root container system such as Docker that lets it use port 80 since it is running as root.

The first involves learning the firewall enough to do this safely, and it doesn't scale to doing such a thing with many containers (a lot of work for each port). The second kind of defeats the point with this trivial web server. The third is one of the best ways to do it, especially since Podman is so compatible with Docker (this is by design).

Saving the container image to disk

Before we can load the container image into Docker, we must first save it to disk since Docker does not use the same local storage. This can be done with Podman's `image save` command, by

```
$ podman image save --format FORMAT -o FILE CONTAINERIMAGE
```

where `FORMAT` is the export format (choose `docker-archive` since the other formats are incompatible with Docker), `FILE` is the name of the output file (it will be an uncompressed tar file (though the layers within it will be compressed tar files)), and `CONTAINERIMAGE` is the `NAME:TAG` or `IMAGE ID` of your container image.

Further Reading

- `podman image save` – `https://docs.podman.io/en/latest/markdown/podman-save.1.html`

Installing Docker

WARNING: Podman and Docker cannot coexist on the CentOS 8 that your VM instances are running. This means that in order to install Docker, you must first uninstall Podman. **DO NOT PROCEED TILL YOU ARE DONE USING PODMAN!**

First, uninstall podman with

```
$ sudo dnf remove podman
```

Docker is not in any of the CentOS 8 default package repositories, nor repositories that can be enabled using packages in the default repositories – just Podman. The easiest way to fix this is to add Docker's official CentOS repository. You can add the repository with

```
$ sudo dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

And then, install the `docker-ce` package. You will be asked if you accept Docker's repository signing key. It should have the fingerprint `060A 61C5 1B55 8A7F 742B 77AA C52F EB6B 621E 9F35`.

Now, Docker is a root container management system that runs as a daemon. It must be started every boot (or enabled so it starts automatically every boot) using `systemctl`:

```
$ sudo systemctl start docker
```

By the way, to enable it for every boot, you would have used `enable` instead of `start`. Other commands are `disable`, `stop`, and `restart` which do exactly what they say.

Further Reading

- Docker installation documentation – `https://docs.docker.com/engine/install/centos/` (note, it uses the older `yum` instead of `dnf`)

Importing the container image into Docker

From this point forward, you need to be careful. Using Docker requires that you run all your commands as root using `sudo`. This means that if you make a typo or aren't careful, you can easily damage your system (e.g. copy a file out of a container and on top of an important file, bind to port 22, etc.).

First, you must import the container image you previously exported to disk into Docker using Docker's `image load` command. Check that the container image was imported using Docker's `image list` command (works just like Podman).

Further Reading

- `docker image list` – https://docs.docker.com/engine/reference/commandline/image_ls/
- `docker image load` – https://docs.docker.com/engine/reference/commandline/image_load/

Running the container image in Docker

Now, run the container but using port 80. Everything is exactly the same as you did in Podman, except you are now using Docker and Docker generally forces you to preceed the names of your container images with `localhost/`. Point your web browser at the VM on port 80 and you should now get a 404 message. You should also see your container if you use Docker's `container list` command.

If you were successful, you can stop the container and remove it.

Further Reading

- `docker container list` – https://docs.docker.com/engine/reference/commandline/container_ls/
- `docker container rm` – https://docs.docker.com/engine/reference/commandline/container_rm/
- `docker container run` – https://docs.docker.com/engine/reference/commandline/container_run/
- `docker container stop` – https://docs.docker.com/engine/reference/commandline/container_stop/

Cleaning up Docker and removing it

To clean up, you need to remove your container image from Docker and then stop Docker itself. You can stop Docker by

```
$ sudo systemctl stop docker.service
```

```
$ sudo systemctl stop docker.socket
```

If you just specify `docker` above, it refers to `docker.service`. You must also stop its socket, hence why there are two invocations.

and uninstall it by

```
$ sudo dnf remove docker-ce
```

Now, you can easily install Podman again, or re-install Docker again.

You can also outright remove the Docker package repository by deleting `/etc/yum.repos.d/docker-ce.repo` if you want.