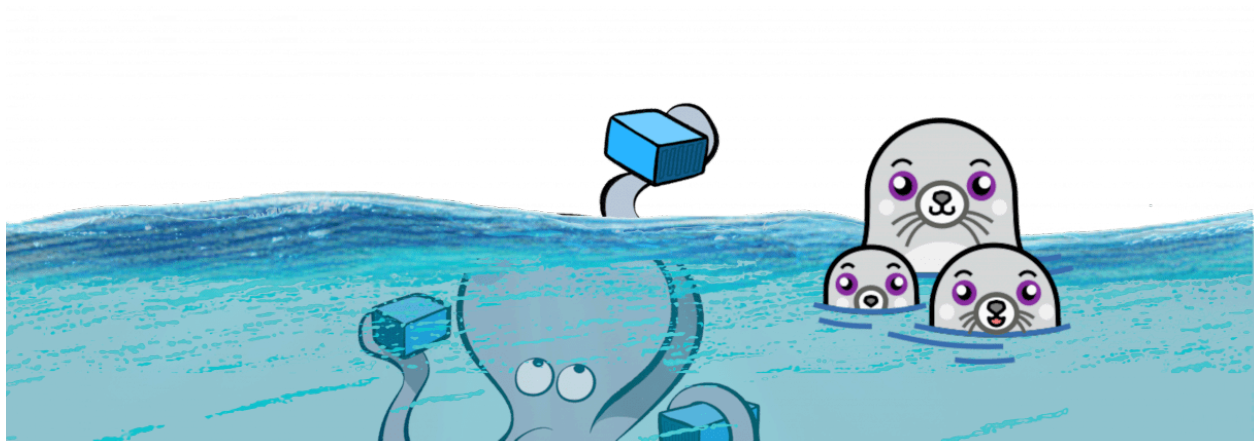
ALLGEMEIN, DATA SCIENCE, EVENTS, NEWS, NEWSTICKER

Using rootless Podman with Docker Compose



Podman is a nice tool, that simplifies running rootless containers without the need for a separate, rootful daemon. One of the tools that you might miss when migrating from Docker to Podman is Docker Compose, which provides an easy-to-use, declarative approach to running and orchestrating containers.

Since the Podman API is compatible with the Docker API, we can use Docker Compose with Podman, with the Podman API simply pretending to be the Docker API when called by Docker Compose. This post shows you how you can get started.

Host setup

On the host, you need three things: Podman, Docker Compose, and a non-root user. Everything else we need for running this example can be found in the Github repository https://github.com/clemens-zauchner/podman-with-docker-compose.

Prerequisites

In this post it is assumed that you have a system user set up already, Podman installed and Docker NOT installed. It is recommended to use a Podman version 4, which ships with RHEL. Furthermore, the code snippets below are assumed to be run as the non-root (system) user.

Installing Docker Compose

You can download the latest version of Docker Compose directly from Github. In this case, we will simply save the binary in the home directory of the non-root user and run it from there. (If the directory *$HOME/bin* does not exist, create it or modify the code below with a directory of your choice.)

Unset

```
wget
https://github.com/docker/compose/releases/latest/download/
docker-compose-linux-$(uname -m) -O
$HOME/bin/docker-compose

$HOME/bin/docker-compose
chmod +x $HOME/bin/docker-compose
export PATH="/home/$USER/bin:$PATH"
```

Starting the Podman API socket

Note: if you are using a system account, you need to export the *XDG_RUNTIME_DIR* to prevent the error *Failed to connect to bus: No medium found* in the next step.

Unset

```
export XDG_RUNTIME_DIR="/run/user/$(id -u)"
```

You can start the rootless Podman API socket via a predefined systemd service using:

Unset

```
systemctl --user start podman.socket
```

Setting the DOCKER_HOST

You need to tell Docker Compose where to find the socket, so it can make API calls.
If you skip this step (and don't have Docker installed, as we assume), you will get an error like this:

Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?

You can fix it by setting the DOCKER_HOST environment variable:

Unset

```
export DOCKER_HOST="unix:///run/user/$(id -u)/podman/podman.sock"
```

The example application

Now let's have a closer look at the files in our example repository. (If you haven't already cloned the repo and you want to follow along, do it now.)

The declarations that Docker Compose interprets can be found in *docker-compose.xml*:

Unset

```
version: "3"
services:
  flask:
    build: .
    ports:
      - 5000:5000
```

We specify one service, called "flask". For the corresponding container image, the build context is the current directory ("."). We also specify to expose the port 5000 on the host, so we can reach the service from the host via *curl* (once the container image has been built and the container is up and running).

Building the image

You can build the image using *docker-compose build*. The build tool for Podman is usually Buildah, but if you use Docker Compose, a BuildKit container is started (however, via the Podman socket, i.e. with Podman as container engine!) and builds the image, as declared in the Dockerfile of our repository.

When you look at the output of *podman ps* right after starting the build process, you can see a container *buildx_buildkit_default* running. This is the container that builds the image.

Starting and stopping the containers

To create and start the container (using the image that just has been built), you can use *docker-compose up*. This command will create a network and start the flask container. The code that runs within the container is a very simple web application defined here. You can verify that everything works as expected using *curl*:

Unset

```
curl localhost:5000 && echo
Hello from Flask!
```

To stop the containers, you can *use docker-compose down*.

Drawbacks

Podman has some features, that are not supported by Docker Compose, e.g. pods or secrets. Although you can define secrets in the *docker-compose.yml*, this is only supported in swarm mode:

Unset

```
version: "3"
services:
  flask:
    build: .
    ports:
      - 5000:5000
    secrets:
      - my_secret

secrets:
  my_secret:
    external: true
```

If you try that, you will get an error message *unsupported external secret mysecret.*

Summary

Podman really does play nicely with Docker Compose. For simple applications or local development, it is certainly an alternative. However, if you want to or need to use more advanced features, you will soon be limited by the features of Docker and Docker Compose.