An Introduction to Podman

Podman, Buildah and Skopeo

HighVail Systems Inc.

Contents

1 An Introduction to Podman

An introduction to using Buildah, Podman and Skopeo to work with containers, and as a replacement for Docker and the Docker CLI

1.1 Workshop Description

In this workshop, we will build a custom application container utilizing the Red Hat Universal Base Image, customize that image and test image functionality. We will then use Skopeo to transfer that image from the local repository to the system repository and configure it to start at boot via Systemd.

Upon completion, we will be able to build images from an existing base image using Buildah and other host based tools. This is just scratching the surface of what can be done with containers and the open source container tooling on RHEL 8.

1.2 Workshop Goal

The goal of this exercise is to create a universal container image with standards compliant and open tools included in Red Hat Enterprise Linux as an alternative to the proprietary Docker tools.

We will show two ways to create the container as alternatives to using Docker.

2 Docker Container Creation

The following is the traditional method for creating a container with Docker, and storing it in a remote repository

2.1 The Dockerfile

The Dockerfile that we will be using for this example is as follows:

```
FROM registry.access.redhat.com/ubi8/ubi:latest
RUN yum -y install httpd
EXPOSE 80
CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]
```

2.2 Building the Image

We will use the Docker `build` command to build the image.

```
$ docker build -t httpd:latest .
[+] Building 24.5s (6/6) FINISHED
 => [internal] load build definition from Dockerfile                          0.0s
 => => transferring dockerfile: 169B                                          0.0s
 => [internal] load .dockerignore                                             0.0s
 => => transferring context: 2B                                               0.0s
 => [internal] load metadata for registry.access.redhat.com/ubi8/ubi:late     6.1s
 => [1/2] FROM registry.access.redhat.com/ubi8/ubi:latest@sha256:82e0fbb     11.0s
 => => resolve registry.access.redhat.com/ubi8/ubi:latest@sha256:82e0fbbf     0.0s
 => => sha256:3269c37eae33b2fc13ef1fceea69f8dd58fa626e008 4.37kB / 4.37kB     0.0s
 => => sha256:d9e72d058dc507f406dc9377495e3d29ce17596f8 74.44MB / 74.44MB     7.4s
 => => sha256:cca21acb641a96561e0cf9a0c1c7b7ffbaaefc92185 1.79kB / 1.79kB     0.3s
 => => sha256:82e0fbbf1f3e223550aefbc28f44dc6b04967fe2578 1.47kB / 1.47kB     0.0s
 => => sha256:8c8dc37fb3c6754572ade71d03a41aab40df98332cc4c1f 737B / 737B     0.0s
 => => extracting sha256:d9e72d058dc507f406dc9377495e3d29ce17596f885c09d0     3.0s
 => => extracting sha256:cca21acb641a96561e0cf9a0c1c7b7ffbaaefc92185bd8a9     0.0s
 => [2/2] RUN yum -y install httpd                                            7.1s
 => exporting to image                                                        0.3s
 => => exporting layers                                                       0.3s
 => => writing image sha256:820282a43deb669bf97d78bc6f05dfb691820570c23e5     0.0s
 => => naming to docker.io/library/httpd:latest                               0.0s

$ docker images
REPOSITORY                          TAG     IMAGE ID      CREATED         SIZE
httpd                               latest  820282a43deb  7 seconds ago   233MB
registry.access.redhat.com/ubi8/ubi latest  3269c37eae33  2 months ago    201MB
```

2.3 Saving the Image to an External Repository

Here we will take that container image and push to an external repository, in this case, Docker hub.

```
$ docker tag httpd:latest aprowse/podman_demo:latest

$ docker push aprowse/podman_demo:latest
```

```
The push refers to repository [docker.io/aprowse/podman_demo]
0caecfe12bc7: Pushed
eb7bf34352ca: Pushed
92538e92de29: Pushed
latest: digest: sha256:a9ecb5d3740f1537f9d1ab5fa2b0524579d0d6fbaa1b28 size: 949
```

3 Open Standards, Open Tools

The next few examples in this exercise will use the open tools available in most distributions. The tools we will use are Podman, Buildah, and Skopeo.

3.1 Overview of buildah, podman and skopeo

3.1.1 Buildah

Specializes in building OCI images. Buildah's commands replicate all of the commands that are found in a Dockerfile. This allows building images with and without Dockerfiles while not requiring any root privileges. The flexibility of building images without Dockerfiles also allows for the integration of other scripting languages into the build process.

3.1.2 Podman

Specializes in all of the commands and functions that help you to maintain and modify OCI images, such as pulling and tagging. It also allows you to create, run, and maintain those containers created from those images.

3.1.3 Skopeo

Specializes in inspecting images and copying images from one location to another, converting formats if necessary.

3.2 Tool Installation

To install them locally, use:

```
$ sudo dnf install -y buildah podman skopeo
Last metadata expiration check: 0:26:44 ago on Tue 09 Feb 2021 09:33:23 AM.
Package podman-2:2.2.1-1.fc33.x86_64 is already installed.
Dependencies resolved.
================================================================================
 Package              Architecture    Version         Repository  Size
================================================================================
Installing:
 buildah              x86_64          1.18.0-1.fc33   updates     7.1 M
 skopeo               x86_64          1:1.2.0-13.fc33 updates     6.1 M

Transaction Summary
================================================================================

Total download size: 13 M
Installed size: 48 M
Downloading Packages:
(1/2): skopeo-1.2.0-13.fc33.x86_64rpm   4.2 MB/s | 6.1 MB     00:01
(2/2): buildah-1.18.0-1.fc33.x86_64.rpm 3.8 MB/s | 7.1 MB     00:01
--------------------------------------------------------------------------------
Total                                           5.7 MB/s |  13 MB     00:02
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing              :                                                 1/1
  Installing             : skopeo-1:1.2.0-13.fc33.x86_64                   1/2
  Installing             : buildah-1.18.0-1.fc33.x86_64                    2/2
  Running scriptlet      : buildah-1.18.0-1.fc33.x86_64                    2/2
  Verifying              : buildah-1.18.0-1.fc33.x86_64                    1/2
  Verifying              : skopeo-1:1.2.0-13.fc33.x86_64                   2/2

Installed:
  buildah-1.18.0-1.fc33.x86_64          skopeo-1:1.2.0-13.fc33.x86_64

Complete!
```

Checking to see if podman is running

```
$ podman ps
CONTAINER ID  IMAGE   COMMAND  CREATED  STATUS  PORTS   NAMES

$ podman run hello-world
Resolved short name "hello-world" to a recorded short-name alias (origin:
    /etc/containers/registries.conf.d/shortnames.conf)
Trying to pull docker.io/library/hello-world:latest...
Getting image source signatures
Copying blob 0e03bdcc26d7 done
Copying config bf756fb1ae done
Writing manifest to image destination
Storing signatures

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

4 The Same Again, but with Open Source Podman

We will run the exact same commands as we did with Docker, but this time with Podman.

4.1 The Dockerfile, Again

The Dockerfile that we used above in the Traditionaal Docker section

```
FROM registry.access.redhat.com/ubi8/ubi:latest
RUN yum -y install httpd
EXPOSE 80
CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]
```

4.2 Building the Image

This time, let's use the `podman build` command to build the image.

```
$ podman build -t httpd:latest .

STEP 1: FROM registry.access.redhat.com/ubi8/ubi:latest
STEP 2: RUN yum -y install httpd
Updating Subscription Management repositories.
Unable to read consumer identity
Subscription Manager is operating in container mode.
Red Hat Enterprise Linux 8 for x86_64 - BaseOS      10 MB/s |   27 MB     00:02
Red Hat Enterprise Linux 8 for x86_64 - AppStre     10 MB/s |   25 MB     00:02
Red Hat Universal Base Image 8 (RPMs) - BaseOS     1.9 MB/s |  772 kB     00:00
Red Hat Universal Base Image 8 (RPMs) - AppStre    6.6 MB/s |  4.9 MB     00:00
Red Hat Universal Base Image 8 (RPMs) - CodeRea     34 kB/s |   13 kB     00:00
Dependencies resolved.
================================================================================
 Package            Arch   Version       Repository                       Size
================================================================================
Installing:
 httpd              x86_64 2.4.37-30    rhel-8-for-x86_64-appstream-rpms  1.4 M
Installing dependencies:
 apr                x86_64 1.6.3-11.el8  rhel-8-for-x86_64-appstream-rpms 125 k
```

```
 <omitted>
  Verifying        : httpd-tools-2.4.37-30.module+el8.3.0+7001+0766b9e7    9/10
  Verifying        : apr-1.6.3-11.el8.x86_64                               10/10
Installed products updated.

Installed:
  apr-1.6.3-11.el8.x86_64
  apr-util-1.6.1-6.el8.x86_64
  apr-util-bdb-1.6.1-6.el8.x86_64
  apr-util-openssl-1.6.1-6.el8.x86_64
  httpd-2.4.37-30.module+el8.3.0+7001+0766b9e7.x86_64
  httpd-filesystem-2.4.37-30.module+el8.3.0+7001+0766b9e7.noarch
  httpd-tools-2.4.37-30.module+el8.3.0+7001+0766b9e7.x86_64
  mailcap-2.1.48-3.el8.noarch
  mod_http2-1.15.7-2.module+el8.3.0+7670+8bf57d29.x86_64
  redhat-logos-httpd-81.1-1.el8.noarch

Complete!
--> f2489417e65
STEP 3: EXPOSE 80
--> d9582b38a9d
STEP 4: CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]
STEP 5: COMMIT httpd:latest
--> 8f12f4299b5
8f12f4299b5d6069b722fe4e77e3bf99b973efea02e77c3f5f4e0e9aea579e40
[aprowse@spanner test]$ podman images
REPOSITORY                          TAG     IMAGE ID      CREATED         SIZE
localhost/httpd                     latest  8f12f4299b5d  7 seconds ago   342 MB
```

4.3 Save to an External Repository

Here we will take that container image and push to an external repository, this time, Red Hat's Quay.

```
$ podman tag httpd:latest quay.io/aprowse/podman_demo:latest

$ podman push quay.io/aprowse/podman_demo:latest
```

```
Getting image source signatures
Copying blob 161c8d157672 done
Copying blob 92538e92de29 skipped: already exists
Copying blob eb7bf34352ca skipped: already exists
Copying config 8f12f4299b done
Writing manifest to image destination
Copying config 8f12f4299b [--------------------------------------] 0.0b / 2.4KiB
Writing manifest to image destination
Writing manifest to image destination
Storing signatures
```

5 Create a Container for Development Using Buildah

In this section, we will use buildah to pull down the Red Hat UBI image to use as our base container image, which we will build on to create our application image.

5.1 Using Buildah to Pull the Base Image

The Red Hat Universal base image (UBI) is a convenient starting point for creating containers. It offers official RHEL bits for building container images, but offers more freedom in how they are used and distributed. There are three versions of it, standard, minimal and runtimes. For this workshop we will use the standard image.

To build an application container from the base image, we will create a working container with buildah. A working container is a temporary container used as the target for buildah commands.

```
$ buildah from registry.access.redhat.com/ubi8/ubi:latest

Getting image source signatures
Copying blob cca21acb641a done
Copying blob d9e72d058dc5 done
Copying config 3269c37eae done
Writing manifest to image destination
Storing signatures
ubi-working-container
```

5.2 Verify the Container Image Pulled Correctly

Verify that the pull request for the container image completed. Using the buildah command will display and allow you to verify what container images your user has access to.

> Buildah will append -working-container to the image name used. If that name already exists, a number will also be appended. For this exercise, you should see an image with the container name ubi-working-container.

```
$ buildah containers

CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME           CONTAINER NAME
7483c71bb84c     *     3269c37eae33 registry..../ub...   ubi-working-container
```

5.3 Install Apache (httpd) on the UBI Base Container Image

The UBI standard variant is very complete, including tools like yum and systemd. You can install httpd via yum in the container using the buildah run subcommand:

```
$ buildah run ubi-working-container -- yum -y install httpd

Updating Subscription Management repositories.
Unable to read consumer identity
Subscription Manager is operating in container mode.

This system is not registered to Red Hat Subscription Management. You can use
   subscription-manager to register.

Red Hat Universal Base  8 (RPMs) - BaseOS           1.4 MB/s | 772 kB    00:00
Red Hat Universal Base  8 (RPMs) - AppStream        6.8 MB/s | 4.9 MB    00:00
Red Hat Universal Base  8 (RPMs) - CodeReady Builder  44 kB/s |  13 kB    00:00
Dependencies resolved.
================================================================================
 Package       Arch    Version                           Repository       Size
================================================================================
Installing:
 httpd         x86_64  2.4.37-30.module+el8.3.0+7001+07  ubi-8-appstream  1.4 M
Installing dependencies
```

```
 apr           x86_64  1.6.3-11.el8                    ubi-8-appstream  125 k
  <omitted>
  Verifying        : httpd-tools-2.4.37-30.module+el8.3.0.x86_64   10/10
Installed products updated.

Installed:
  apr-1.6.3-11.el8.x86_64
  apr-util-1.6.1-6.el8.x86_64
  apr-util-bdb-1.6.1-6.el8.x86_64
  apr-util-openssl-1.6.1-6.el8.x86_64
  httpd-2.4.37-30.module+el8.3.0+7001+0766b9e7.x86_64
  httpd-filesystem-2.4.37-30.module+el8.3.0+7001+0766b9e7.noarch
  httpd-tools-2.4.37-30.module+el8.3.0+7001+0766b9e7.x86_64
  mailcap-2.1.48-3.el8.noarch
  mod_http2-1.15.7-2.module+el8.3.0+7670+8bf57d29.x86_64
  redhat-logos-httpd-81.1-1.el8.noarch

Complete!
```

This subcommand acts like the RUN directive in an OCI file. Since the yum command includes a switch, we need to use the -- syntax to tell buildah run there are no buildah options to look for past this point. This is not buildah specific, but applies to bash when passing commands.

5.4 Install a Simple Home Page

Once the packages are installed in the working container, place a one-line home page we can use to check that our container works properly.

```
$ echo 'Buildah welcomes you to my RHEL8 container!' > index.html

$ buildah copy ubi-working-container index.html /var/www/html/index.html

ba03d144c358eeffe56a50d5058aef8a90f3b3d34cf115df748f4a727d5ab756
```

5.5 Set `httpd` to Start at Launch

Instead of using init scripts, the webserver will be started directly when the container is started. In order to keep the container running while the webserver is up, the foreground flag is added or

the container would end as soon as it goes into the background. To set `httpd` to start when the container is run, modify the metadata with the buildah `config` command.

```
$ buildah config --cmd "/usr/sbin/httpd -D FOREGROUND" ubi-working-container
```

The above option to buildah config is equivalent to the CMD directive in an OCIFile.

5.6 Expose the `http` Port on the Container

To get access to the web server, http port 80 needs to be opened

```
$ buildah config --port 80 ubi-working-container
```

5.7 Commit the Changes to the Modified Base Container Using `buildah`

Once the contents of the working container are complete, and the metadata has been updated, save the working container as the target application image using `buildah commit`. During the container customization process, you can choose how often you want to save your customizations in order to test each modification that has been completed. In this case we are saving both the installation of apache, a simple home page and the directive to start the httpd service:

```
$ buildah commit ubi-working-container httpd

Getting image source signatures
Skipping fetch of repeat blob sha256:24d85c895b6b870f6b84327a5e3
Skipping fetch of repeat blob sha256:c613b100be1645941fded703dd6
Skipping fetch of repeat blob sha256:188ab351dfda8afc656a38073df
Copying blob sha256:8df24355b15ad293a5dd60d0fe2c14dca68b0412b62f
26.80 MiB / 26.80 MiB [====================================] 0s
Copying config sha256:b04fe2c73b034e657da2fee64c340c56086a382657
2.42 KiB / 2.42 KiB [======================================] 0s
Writing manifest to image destination
Storing signatures
B04fe2c73b034e657da2fee64c340c56086a38265777556fa8a02c5f12896e66
```

In this example, each previous Buildah command results in a separate layer, much like building using an OCIFile. Note that we have named our save point as *httpd*. You can change this to any label that will reflect what changes you have made at that given save point.

5.8 Use Podman to Inspect Available Images

In the previous steps we used Buildah to pull down a new image and customize that image. The last step of the section had us commit the changes to the container and name it httpd. Using the `podman` command, we can view what containers are available to start and run.

```
$ podman images
REPOSITORY                            TAG     IMAGE ID     CREATED        SIZE
localhost/httpd                       latest  6b5abf0be18f 16 seconds ago 242 MB
registry.access.redhat.com/ubi8/ubi   latest  3269c37eae33 2 months ago   208 MB
```

> The name matches what was set using `buildah commit`.

5.9 Use Podman to Start the Customized Container and Bind Port 8080

Podman and buildah use the same local image storage locations, which lets us immediately run our new image without specifying the location of the container or system on which the container will run. Note we are using the name *httpd* that we created in our previous section. As mentioned previously, you can launch, test, and then stop the container as you make each individual change. This can be used for general application testing or debugging of a change made to the container during customization with buildah.

The container's port 80 is at this point bound to port 8080 so it could be started by a non-root user.

```
$ podman run -d -p 8080:80 httpd

f4d9db69e9b512517f9490d3bcc5096e69cca5e9b3a50b3890430da39ae46573
```

5.10 Inspect Container and Verify the Application in the Container is Running and Accessible

Now, we can check the status of the application container using podman. Note you can also see the forwarded ports:

```
$ podman ps

CONTAINER ID  IMAGE                   COMMAND              CREATED         ST>
f4d9db69e9b5  localhost/httpd:latest  /usr/bin/run-http... 16 seconds ago  Up>
```

Further, you can view the container's processes with the following:

```
$ podman top -l

USER     PID  PPID %CPU    ELAPSED         TTY   TIME   COMMAND
root     1    0    0.000   34.315182881s   ?     0s     /usr/sbin/httpd -D FOR>
apache   7    1    0.000   34.315664108s   ?     0s     /usr/sbin/httpd -D FOR>
apache   8    1    0.000   34.315820402s   ?     0s     /usr/sbin/httpd -D FOR>
apache   9    1    0.000   34.315883741s   ?     0s     /usr/sbin/httpd -D FOR>
apache   10   1    0.000   34.31593985s    ?     0s     /usr/sbin/httpd -D FOR>
```

Now, we can test retrieval of our example home page:

```
$ curl -s http://localhost:8080

Buildah welcomes you to my RHEL8 container!
```

> Note the URL specified matches the port mapping specified on the podman run command.

5.11 Stopping the Container

Since your test was successful, you can now stop the container, and continue with additional customization that you would like to try out. Remember to commit your changes as often as you would like, during the customization process, and use names that reflect the customization you have done to ease troubleshooting.

```
$ podman stop -a

f4d9db69e9b512517f9490d3bcc5096e69cca5e9b3a50b3890430da39ae46573
```

This will stop all containers that you have running via podman.

You can verify that the container has stopped running by looking at the list of container processes:

```
$ podman ps -la
CONTAINER ID  IMAGE                    COMMAND               STATUS            >
f4d9db69e9b5  localhost/httpd:latest   /usr/sbin/httpd -...  Exited (0) 7 secon>
```

> Notice the STATUS field is now reported as Exited.

Alternatively, if you would prefer to stop only a single container, you can utilize `podman ps` to identify the Container ID you wish to stop. (If you've already performed the `stop -a`, you can re-start the container with the `podman run` command shown above.) Then use the following command, with your unique Container ID number, to shutdown a single instance.

For example:

```
podman stop f4d9db69e9b5
```

6 Use Skopeo and Podman to Integrate the Container into Systemd

Running as a regular user, the container work that you have done is stored in your home directory. We will move it to the system image store in `/var/lib/`, enable it and start the application.

6.1 Inspecting the *httpd* Image

First let's use Skopeo to inspect the image.

```
$ skopeo inspect containers-storage:localhost/httpd

{
    "Name": "localhost/httpd",
    "Digest": "sha256:0dbc14b4aa06a3232087d5fa329b158dfe580686fa00e9383f78eef",
    "RepoTags": [],
    "Created": "2021-01-12T20:00:34.021305317Z",
    "DockerVersion": "",
    "Labels": {

<<output truncated>>

}
```

6.2 Transfer the Image into the Operating System Image Store

First export the image from the users image store into an archive file. Skopeo can export containers into either docker archive or OCI archive if we want to put the container into a file. Using the OCI archive format:

```
$ skopeo copy containers-storage:localhost/httpd oci-archive:httpd.tar
Getting image source signatures
Copying blob 226bfaae015f done
Copying blob 70056249a0e2 done
Copying blob 1ff90c7e6397 done
Copying config 80dd2eb93b done
Writing manifest to image destination
Storing signatures
```

Import the archive into the system image store

```
$ sudo skopeo copy oci-archive:httpd.tar containers-storage:localhost/httpd

WARN[0000] Not using native diff for overlay, this may cause degraded
     performance for building images: kernel has CONFIG_OVERLAY_FS_REDIRECT_DIR
    enabled
Getting image source signatures
Copying blob b80ee16c8662 done
Copying blob 6eeb9b4a640f done
Copying blob ae48556e82ac done
Copying config 80dd2eb93b done
Writing manifest to image destination
Storing signatures
```

The container should now be visible in the system image store

```
$ sudo podman images

REPOSITORY        TAG      IMAGE ID       CREATED          SIZE
localhost/httpd   latest   80dd2eb93b53   37 minutes ago   242 MB
```

6.3 Copy the Image to the Remote Quay Repo

To run an image from another host, say an OpenShift cluster, we need to make that image avaialble. To do that, we use skopio to copy th elocal image to quay.io.

> Note: This can be done through podman as well with the `tag` and `push` options.

```
$ skopeo copy containers-storage:localhost/httpd:latest \
     docker://quay.io/aprowse/podman_demo:1.0

Getting image source signatures
Copying blob 3947a649dae1 skipped: already exists
Copying blob eb7bf34352ca skipped: already exists
Copying blob 92538e92de29 [--------------------------------------] 0.0b / 0.0b
Copying config 57443b6103 [--------------------------------------] 0.0b / 2.2KiB
Writing manifest to image destination
Copying config 57443b6103 [--------------------------------------] 0.0b / 2.2KiB
Writing manifest to image destination
Writing manifest to image destination
Storing signatures
```

7 Exporting a Container Definition for use in OpenShift

If you've built and tested a container with podman, and are happy with the results, you can very easily share that container with OpenShift.

7.1 Generating a Kubernetes Manifest

Podman can generate a kubernetes manifest with a single command, as seen below. This functionality is still in beta, so there may be a little tweaking involved. At least the hard work is done for you.

We get the running container with the `podman ps --quiet -l` command, or we could specify it if we have multipe active containers running

```
$ podman generate kube $(podman ps --quiet -l) > export.yaml
```

Looking at the manifest created.

```
$ cat export.yaml

# Generation of Kubernetes YAML is still under development!
#
# Save the output of this file and use kubectl create -f to import
# it into Kubernetes.
#
# Created with podman-2.2.1
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2021-02-11T15:07:56Z"
  labels:
    app: web         # <== change the app name to something better
  name: webapp       # <== change the name to something better
spec:
  containers:
  - command:
    - /usr/sbin/httpd
    - -D
    - FOREGROUND
    env:
    - name: PATH
      value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
    - name: TERM
      value: xterm
    - name: container
      value: oci
    - name: HOSTNAME
    image: quay.io/aprowse/podman_demo:1.0 # <== change to this from local/httpd:latest
    name: condescendingpike
    ports:
    - containerPort: 80
      hostPort: 8080
      protocol: TCP
    resources: {}
    securityContext:
```

```
      allowPrivilegeEscalation: true
      capabilities:
        drop:
        - CAP_MKNOD
        - CAP_NET_RAW
        - CAP_AUDIT_WRITE
      privileged: false
      readOnlyRootFilesystem: false
      seLinuxOptions: {}
    workingDir: /
status: {}
---
kind: Service # <== Add this, it's missing
metadata:
  creationTimestamp: null
spec: {}
status:
  loadBalancer: {}
```

7.2 Importing the Manifest into OpenShift

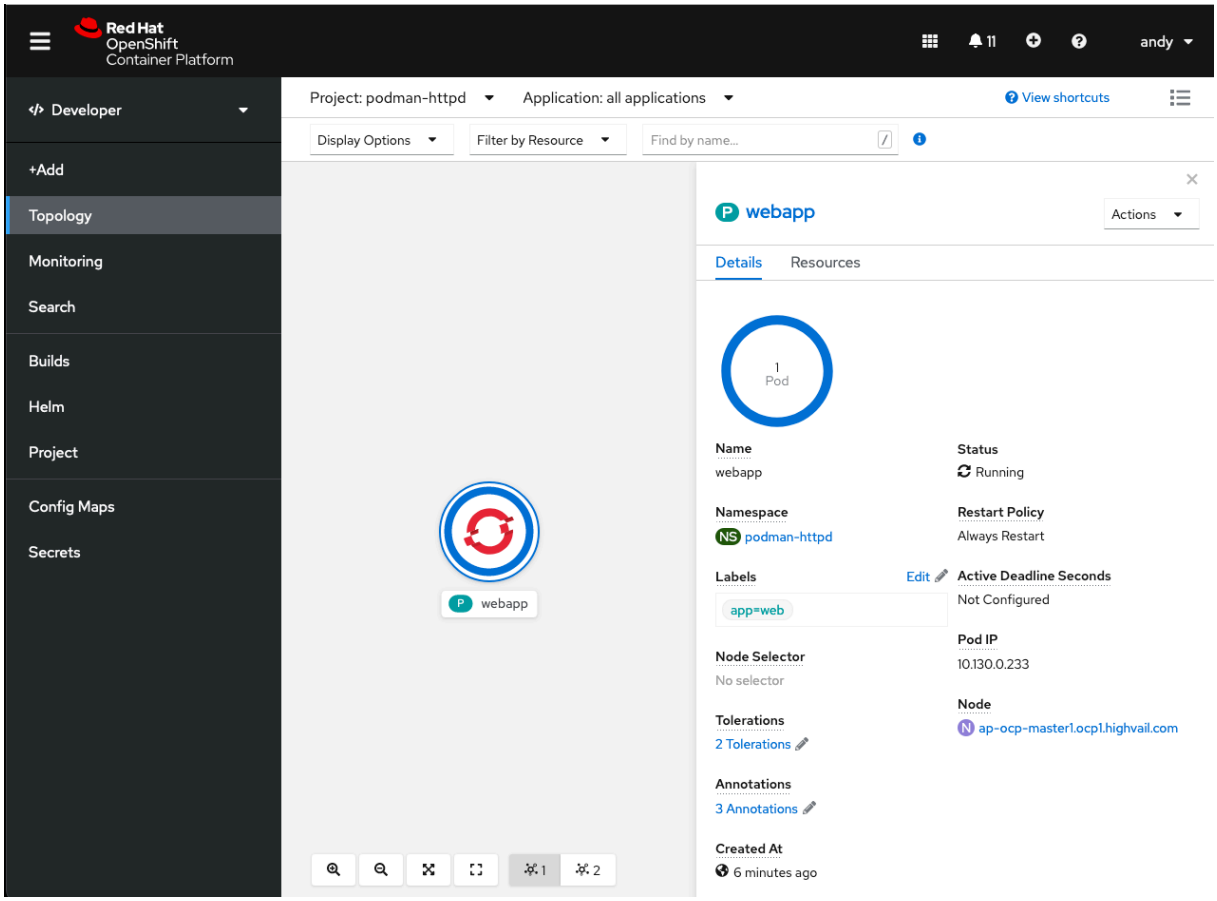
This manifest can deployed to Kubernetes. In this example we will deploy to OpenShift.

```
$ oc new-project podman-httpd

$ oc create -f export.yaml

pod/podman-httpd created
error: unable to decode "export.yaml": Object 'Kind' is missing in '{"metadata"
  :{"creationTimestamp":null},"spec":{},"status":{"loadBalancer":{}}}'

$ $ oc describe pod/webapp
Name:         webapp
Namespace:    podman-httpd
Priority:     0
Node:         ap-ocp-master1.ocp1.highvail.com/10.10.12.11
Start Time:   Thu, 11 Feb 2021 10:46:57 -0500
```

```
Labels:       app=web
Annotations:  k8s.v1.cni.cncf.io/network-status:
                [{
                    "name": "",
                    "interface": "eth0",
                    "ips": [
                        "10.130.0.233"
                    ],
                    "default": true,
                    "dns": {}
                }]
              k8s.v1.cni.cncf.io/networks-status:
                [{
                    "name": "",
                    "interface": "eth0",
                    "ips": [
                        "10.130.0.233"
                    ],
                    "default": true,
                    "dns": {}
                }]
              openshift.io/scc: node-exporter
Status:       Running
IP:           10.130.0.233
IPs:
  IP:  10.130.0.233
Containers:
  condescendingpike:
    Container ID:  cri-o://1328a3d8d73a1dba4c5a551b8840ca8e85e7a366d02eac4c7d0>
    Image:         quay.io/aprowse/podman_demo:1.0
    Image ID:      quay.io/aprowse/podman_demo@sha256:5f8bf53c13fc4555d35f1c63>
    Port:          80/TCP
    Host Port:     8080/TCP
    Command:
      /usr/sbin/httpd
      -D
      FOREGROUND
```

```
    State:          Running
      Started:      Thu, 11 Feb 2021 10:47:06 -0500
    Ready:          True
    Restart Count:  0
    Environment:
      PATH:       /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      TERM:       xterm
      container:  oci
      HOSTNAME:
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-6cx (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-6nscx:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-6nscx
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason          Age    From               Message
  ----    ------          ----   ----               -------
  Normal  Scheduled       2m50s  default-scheduler  Successfully assigned podm>
  Normal  AddedInterface  2m49s  multus             Add eth0 [10.130.0.233/23]
  Normal  Pulling         2m49s  kubelet            Pulling image "quay.io/apr>
  Normal  Pulled          2m43s  kubelet            Successfully pulled image >
  Normal  Created         2m42s  kubelet            Created container condesce>
  Normal  Started         2m42s  kubelet            Started container condesce>
```

And the pod in OpenShift



This is an example of a single container export, but you can export complete pods as well.

8 Appendix

8.1 Some Commonly Used Commands

If a container will no longer be used, you can remove it from the system using `podman rm`. In the command below, we use a bit of bash scripting to return the CONTAINER ID of the last container that was running as it is unique to each container image.

```
$ podman rm $(podman ps --quiet -l)

af2d3774f20b5afb4505a4eb3fea20df5861afd6ec06b9271b6419ff1515106d
```

The output of this removal is the full CONTAINER ID which was removed from the system.