

# Containers Demystified

A hands-on approach to understanding container mechanics, standards, and tooling.  
Includes Docker, Podman, and Buildah

## Class Labs

Version 1.4 by Brent Laster for Tech Skills Transformations LLC

11/23/2022

**Important Prereq:** These labs assume you have a system ready with Git and Docker installed and a username/password setup on hub.docker.com. You should also have run the prepull-image script ahead of the workshop if possible. If you have not done that, please complete the steps before continuing

## Lab 1- Creating Images

**Purpose:** In this lab, we'll see how to do basic operations like building images.

1. If you haven't already, clone down the ctr-de repository from GitHub.

```
$ git clone https://github.com/skillrepos/ctr-de
```

2. Switch into the directory for our work.

```
$ cd ctr-de
```

3. Do an ls command and take a look at the files that we have in this directory.

```
$ ls
```

4. Take a moment and look at each of the files that start with "Dockerfile". See if you can understand what's happening in them.

```
$ cat Dockerfile_roar_db_image
$ cat Dockerfile_roar_web_image
```

5. Now let's build our docker database image. Type (or copy/paste) the following command: (Note that there is a space followed by a dot at the end of the command that must be there.)

```
$ docker build -f Dockerfile_roar_db_image -t roar-db .
```

- Next build the image for the web piece. This command is similar except it takes a build argument that is the war file in the directory that contains our previously built webapp.

(Note the space and dot at the end again.)

```
$ docker build -f Dockerfile_roar_web_image --build-arg warFile=roar.war -t roar-web .
```

- Now, let's tag our two images with your user name for the docker.io or quay.io repositories. We'll give them a tag of "v1" as opposed to the default tag that Docker provides of "latest".

```
$ docker tag roar-web <your registry username>/roar-web:0.0.1
$ docker tag roar-db <your registry username>/roar-db:0.0.1
```

- Do a docker images command to see the new images you've created.

```
$ docker images | grep roar
```

- Now let's actually run our web image as a container. To do this we'll run the container and expose a port to view it on. Execute the following command:

```
$ docker run -p 8088:8080 <your registry username>/roar-web:0.0.1
```

- Now you can open up a browser session at the port we exposed and see the webapp running. Open the URL below in a browser.

<http://localhost:8088/roar>



- You'll notice that there is no data showing up in the table. That's because our database container is not running and being accessed. We'll see how to fix that in the next lab. In the meantime, you can stop the webapp container from running via Ctrl-C.

**END OF LAB**

## Lab 2 – Making images work together

**Purpose:** In this lab, we'll see how to make multiple containers execute together with docker compose and use the docker inspect command to get information to see our running app.

1. Take a look at the docker compose file for our application and see if you can understand some of what it is doing.

```
$ cat docker-compose.yml
```

2. Run the following command to compose the two images together that we built in lab 1.

```
$ docker-compose up
```

3. You should see the different processes running to create the containers and start the application running. Take a look at the running containers that resulted from this command.

Note: We'll leave the processes running in the first session, so **open a second command prompt/terminal emulator** and enter the command below.

```
$ docker ps | grep roar
```

4. Make a note of the first 3 characters of the container id (first column) for the web container (row with **roar-web** in it). You'll need those for the next step.
5. Let's find the web address so we can look at the running application. To do this, we will search for the information via a docker **inspect** command. Enter this command in the **second** terminal session, substituting in the characters from the container id from the step above for "<container id>" - the one for *roar-web*.

(For example, if the line from docker ps showed this:

```
237a48a2aeb8    roar-web    "catalina.sh run"    About a minute ago Up About a minute 0.0.0.0:8089->8080/tcp
```

then <container id> could be "**237**". Also note that "IPAddress" is case-sensitive.)

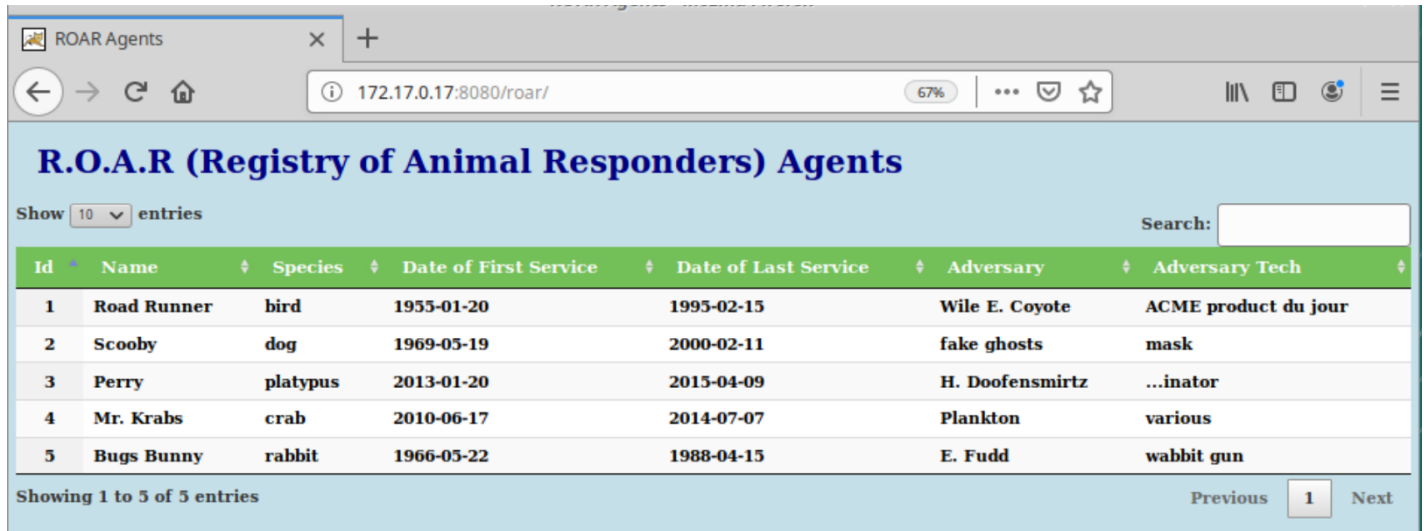
Make a note of the url that is returned.

```
$ docker inspect <container id> | grep IPAddress
```

- Open a web browser and go to the url below, substituting in the ip address from the step above for "<ip address>". (Note the :8080 part added to the ip address)

<http://<ip address>:8089/roar/>

- You should see the running app on a screen like the following:



Id	Name	Species	Date of First Service	Date of Last Service	Adversary	Adversary Tech
1	Road Runner	bird	1955-01-20	1995-02-15	Wile E. Coyote	ACME product du jour
2	Scooby	dog	1969-05-19	2000-02-11	fake ghosts	mask
3	Perry	platypus	2013-01-20	2015-04-09	H. Doofensmirtz	...inator
4	Mr. Krabs	crab	2010-06-17	2014-07-07	Plankton	various
5	Bugs Bunny	rabbit	1966-05-22	1988-04-15	E. Fudd	wabbit gun

- Now let's find out some more details about these containers. For these commands, you can use the partial ids of either of the two containers. First, look at the top processes in this container. Then, look at the stats and note how much CPU the container is using. Ctrl-C out of that when you're done.

```
$ docker top <container id>
$ docker stats <container id>
```

- Now pause the same container's processes and look at the stats again to see the CPU usage drop off. You can also refresh the browser to see what happens when we have the container paused. Again you can Ctrl-C when done.

```
$ docker pause <container id>
$ docker stats <container id>
```

- When done, you can Ctrl-C again to stop the stats process and then unpause the container. If you want, you can view stats again to see the CPU use go back to normal.

```
$ docker unpause <container id>
(optional) $ docker stats <container id>
```

END OF LAB

### Lab 3 – Debugging Docker Containers

**Purpose:** Understanding commands and steps that we can use to learn more about what's going on in our containers if we run into problems.

1. Let's get a description of all of the attributes of our containers. For these commands, use the same 3 character container id you used in step 2.

Run the inspect command. Take a moment to scroll around the output.

```
$ docker inspect <container id>
```

2. Now, let's look at the logs from the running container. Scroll around again and look at the output.

```
$ docker logs <container id>
```

3. While we're at it, let's look at the history of the image (not the container).

```
$ docker history roar-web
```

4. Now, let's suppose we wanted to take a look at the actual database that is being used for the app. This is a mysql database but we don't have mysql installed on the VM. So how can we do that? Let's connect into the container and use the mysql version within the container. To do this we'll use the "docker exec" command. First find the container id of the db container.

```
$ docker ps | grep roar-db
```

5. Make a note of the first 3 characters of the container id (first column) for the db container (row with **roar-db** in it). You'll need those for the next step.
6. Now, let's exec inside the container so we can look at the actual database.

```
$ docker exec -it <container id> bash
```

Note that the last item on the command is the command we want to have running when we get inside the container – in this case the bash shell.

7. Now, you'll be inside the db container. Check where you are with the `pwd` command and then let's run the `mysql` command to connect to the database. (Type these at the `/#` prompt. Note no spaces between the options `-u` and `-p` and their arguments. You need only type the part in **bold**.)

```
root@container-id:/# pwd  
root@container-id:/# mysql -uadmin -padmin registry
```

(Here `-u` and `-p` are the userid and password respectively and `registry` is the database name.)

8. You should now be at the "`mysql>`" prompt. Run a couple of commands to see what tables we have and what is in the database. (Just type the parts in **bold**.)

```
mysql> show tables;  
mysql> select * from agents;
```

9. Exit out of `mysql` and then out of the container.

```
mysql> exit  
root@container-id:/# exit
```

10. Since we no longer need our docker containers running or the original images around, let's go ahead and get rid of them with the commands below.

(Hint: `docker ps | grep roar` will let you find the ids more easily)

Stop the containers

```
$ docker stop <container id for roar-web>  
$ docker stop <container id for roar-db>
```

Remove the containers

```
$ docker rm <container id for roar-web>  
$ docker rm <container id for roar-db>
```

END OF LAB

## Lab 4: Mapping Docker images and containers with the filesystem

**Purpose:** In this lab, we'll explore how layers, images and containers are actually mapped and stored in the filesystem.

**Suggested setup:** Install the "jq" tool if you don't have it from <https://stedolan.github.io/jq/>

1. First, we need to access the underlying storage area for Docker. If you are running Docker on a Linux machine, you can open a terminal session to "/var/lib/docker".

If you are on a Windows or Mac system and have Docker Desktop installed, run the following command in a terminal.

```
$ docker run -it --privileged --pid=host debian nsenter -t 1 -m -u -n -i sh
```

Now you should be able to change to the /var/lib/docker directory and see the files in that structure.

2. In another terminal session, let's run an interactive container based off of Ubuntu.

```
$ docker run -ti ubuntu:18.04 bash
```

3. After pulling down an instance of the image, it will be started running for you and you'll be inside the image. Let's make some simple changes so we can see how these are represented and stored in the underlying file system. We'll delete one file, create a second one and then exit the container.

```
# rm /etc/environment
# echo new > /root/newfile.txt
# exit
```

4. Find the first 4 characters of the ubuntu container you were working with. You can either get it from the previous steps or you can use a command like the one below to find it.

```
$ docker ps -a | grep ubuntu
```

5. Now do a docker diff command to summarize the differences. After this step, we'll track those changes back through to the file system.

```
$ docker diff <first 4 characters of container id>
```

6. Run a docker inspect command to find the underlying filesystem directories for the layers - using the first 4 characters from the container id and the jq tool to get the "graphdriver" data.

```
$ docker inspect <first 4 chars of container id> | jq '.[0].GraphDriver.Data'
```

7. You should see output like the following. Take note of the value for "UpperDir". Select that and copy it.

```
{
  "LowerDir":
"/var/lib/docker/overlay2/c19f1aa7797551da6701cfe5bb716665189d7191e4bc27ba503e6eb1d3a864cc-init/diff:/var/lib/docker/overlay2/4d037a0e2bb0f50d031382246c8374382fdd126b57960ff99d4b4c9be04cffd2/diff",
  "MergedDir":
"/var/lib/docker/overlay2/c19f1aa7797551da6701cfe5bb716665189d7191e4bc27ba503e6eb1d3a864cc/merged",
  "UpperDir":
"/var/lib/docker/overlay2/c19f1aa7797551da6701cfe5bb716665189d7191e4bc27ba503e6eb1d3a864cc/diff",
  "WorkDir":
"/var/lib/docker/overlay2/c19f1aa7797551da6701cfe5bb716665189d7191e4bc27ba503e6eb1d3a864cc/work"
}
```

8. In the other terminal window, where you are in the `/var/lib/docker` directory, do an "ls" of that directory to see what's in the Docker filesystem location.

```
$ ls <UpperDir path value copied from previous step>
```

The results should look something like this - showing the two top directories.

```
/ # ls -la
/var/lib/docker/overlay2/c19f1aa7797551da6701cfe5bb716665189d7191e4bc27ba503e6eb1d3a864cc/diff

total 16
drwxr-xr-x  4 root    root    4096 Nov 23 21:28 .
drwx--x---  4 root    root    4096 Nov 23 21:38 ..
drwxr-xr-x  2 root    root    4096 Nov 23 21:30 etc
drwx-----  2 root    root    4096 Nov 23 21:30 root
```

9. Now, take a look at the "etc" directory and you should see the file that was removed.

```
$ ls -la <UpperDir path value copied from previous step>/etc
```

The results should look something like this showing the removed file (as seen on windows). Note the "c" attribute marking this as a removed/masking file.

```
/ # ls -la
/var/lib/docker/overlay2/c19f1aa7797551da6701cfe5bb716665189d7191e4bc27ba503e6eb1d3a864cc/diff/etc
total 8
drwxr-xr-x  2 root    root    4096 Nov 23 21:30 .
drwxr-xr-x  4 root    root    4096 Nov 23 21:28 ..
c-----  1 root    root      0,  0 Nov 23 21:30 environment
```



10. Next, look at the "root" directory and you should see the file that was created.

```
$ ls -la <UpperDir path value copied from previous step>/root
```

The results should look something like this showing the added file.

```
/ # ls
/var/lib/docker/overlay2/c19f1aa7797551da6701cfe5bb716665189d7191e4bc27ba503e6eb1d3a864cc/diff/root
```

```
total 16
drwx----- 2 root    root    4096 Nov 23 21:30 .
drwxr-xr-x  4 root    root    4096 Nov 23 21:28 ..
-rw-----  1 root    root     54 Nov 23 21:30 .bash_history
-rw-r--r--  1 root    root     4 Nov 23 21:30 newfile.txt
```

11. If you want to see where the original image is stored, grab the second path under the "LowerDir" section (after the "init/diff:" piece). It is highlighted below.

```
{
  "LowerDir":
    "/var/lib/docker/overlay2/c19f1aa7797551da6701cfe5bb716665189d7191e4bc27ba503e6eb1d3a864cc-
    init/diff:/var/lib/docker/overlay2/4d037a0e2bb0f50d031382246c8374382fd126b57960ff99d4b4c9be04cffd2/diff",
  "MergedDir":
    "/var/lib/docker/overlay2/c19f1aa7797551da6701cfe5bb716665189d7191e4bc27ba503e6eb1d3a864cc/merged",
  "UpperDir":
    "/var/lib/docker/overlay2/c19f1aa7797551da6701cfe5bb716665189d7191e4bc27ba503e6eb1d3a864cc/diff",
  "WorkDir":
    "/var/lib/docker/overlay2/c19f1aa7797551da6701cfe5bb716665189d7191e4bc27ba503e6eb1d3a864cc/work"
}
```

12. You can do an "ls" on the path copied from the previous step and you'll see the original starting layer for the container. You can also look at the "etc" and "root" directories to see the original state of those without the changes we made.

```
$ ls -la <path value from 2nd part of LowerDir copied from previous step>
```

```
$ ls -la <path value from 2nd part of LowerDir copied from previous step>/root
```

```
$ ls -la <path value from 2nd part of LowerDir copied from previous step>/etc
```

END OF LAB

## Lab 5 - Working with Podman

**Purpose:** In this lab, we'll get a chance to work with Podman, an alternative to Docker that also includes the abilities to group and work with containers in "pods".

1. For this lab, you can install podman directly if you want. If you haven't already, via the instructions at <https://podman.io/getting-started/installation>. If you go that route, and you are on a mac or Windows system, you will need to follow the instructions to get the podman virtual machine setup and then do

```
$ podman machine init
$ podman machine start
```

2. OR you can run podman via a container or via the podman virtual machine. The command to run via a container is shown below.

```
$ docker run -it -p 127.0.0.1:8087:8087 --privileged -v <working
dir>:/work quay.io/podman/stable bash
```

3. Check that podman is installed and responding.

```
$ podman version
```

4. Now that you have podman installed, change into the directory with the docker content.

```
$ cd /work/ctr-de
```

4. Now, build the two images (the web one and the database one) that we need for our application. Note that the syntax for podman is just like the syntax for Docker. Afterwards, you can see the images with podman.

```
$ podman build -t roar-web:1.0.0 --build-arg warFile=roar.war -f
Dockerfile_roar_web_image .
```

```
$ podman build -t roar-db:1.0.0 -f Dockerfile_roar_db_image .
```

```
$ podman images
```

5. Now let's create a pod.

```
$ podman pod create --name roar-pod -p 8087:8080 --network bridge
```

6. Next, we'll list the pod we have and then inspect it to look at it closer.

```
$ podman pod ls
```

```
$ podman inspect roar-pod
```

7. Notice the inspect lists one container at the bottom. Let's look closer at what that container is.

```
$ podman ps -a --pod
```

8. Add the web image as a container to the pod.

```
$ podman run --pod roar-pod --name roar-web --ipc=private -d roar-web:1.0.0
```

9. Finally, we'll add the database image as a container to the pod.

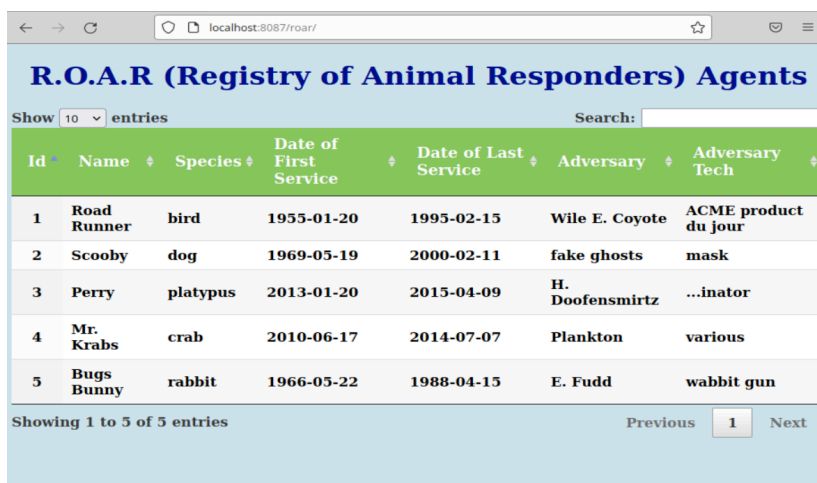
```
$ podman run --pod roar-pod --name roar-db --env-file env.list --ipc=private -d roar-db:1.0.0
```

10. You can now see the containers running in the pod.

```
$ podman inspect roar-pod
```

11. Now you can open up the url below in a browser and see the application running.

<http://localhost:8087/roar>



Id	Name	Species	Date of First Service	Date of Last Service	Adversary	Adversary Tech
1	Road Runner	bird	1955-01-20	1995-02-15	Wile E. Coyote	ACME product du jour
2	Scooby	dog	1969-05-19	2000-02-11	fake ghosts	mask
3	Perry	platypus	2013-01-20	2015-04-09	H. Doofensmirtz	...inator
4	Mr. Krabs	crab	2010-06-17	2014-07-07	Plankton	various
5	Bugs Bunny	rabbit	1966-05-22	1988-04-15	E. Fudd	wabbit gun

END OF LAB

## Lab 6: Working with Buildah

**Purpose:** In this lab, we'll work with the container build and management tool Buildah.

1. If you are running on a Linux system or VM, you can follow instructions at <https://github.com/containers/buildah/blob/main/install.md> to install Buildah. If you are running with Docker Desktop on a Mac or Windows system, you can run it via a Docker container.

2. When you run the Docker container, mount the area where you have the ctr-de directory available - indicated by the <working dir> in step 3.

3. Run the command below to access buildah via a container.

```
$ docker run -it --device /dev/fuse:rw --privileged -v <working  
dir>:/work quay.io/buildah/stable bash
```

4. If you're running in a container, you'll now be in the container with access to buildah. Go to the work directory that you mounted into the container. And run the command below to produce new images using the bash script. After the images are created, you should be able to see them via the "buildah images" command.

```
$ cd /work
```

5. Now, run the bash script that will use buildah to build the images instead of the Dockerfiles. Also, we need to pass in the built deliverable to be pulled in for the webapp. That's what roar.war is.

```
$ bash ./buildah-roar.sh docker test roar.war  
$ buildah images
```

6. For this next step, you will need your docker.io userid or your quay.io userid. Login to one of these using the buildah login command and your username/password.

```
$ buildah login docker.io -OR- $ buildah login quay.io
```

7. After logging in to the registry, tag your images with your username appropriately.

```
$ buildah tag roar-web:1.0.1 <userid>/roar-web:1.0.1  
$ buildah tag roar-db:1.0.1 <userid>/roar-db:1.0.1
```

8. Push the images out to the registry.

```
$ buildah push <userid>/roar-web:1.0.1
```

```
$ buildah push <userid>/roar-db:1.0.1
```

9. Now you can switch back to where podman is running.

10. Use podman to pull the updated db image you just pushed.

```
$ podman pull <userid>/roar-db:1.0.1
```

11. Take a look at the containers you have locally and note the ids of the roar-db one you just pulled and the previous

```
$ podman container list --all | grep roar-db:1.0.0
```

12. Now, we'll use podman to remove the old container from the pod and replace it with the new one.

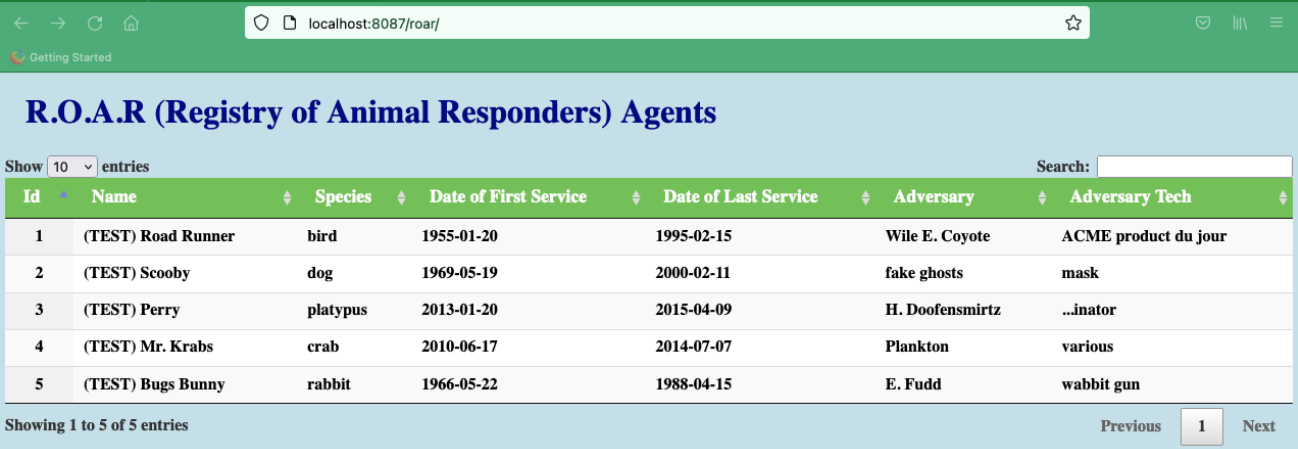
```
$ podman container stop <first 4 chars of container 1.0.0 name>
```

```
$ podman container rm <first 4 chars of container 1.0.0 name>
```

13. Now, we'll use podman to add the new container into the pod.

```
$ podman run --pod roar-pod --env-file env.list --name roar-db --  
ipc=private -d <userid>/roar-db:1.0.1
```

14. Refresh the application in the browser and you should see a version of the app running with test data.



Id	Name	Species	Date of First Service	Date of Last Service	Adversary	Adversary Tech
1	(TEST) Road Runner	bird	1955-01-20	1995-02-15	Wile E. Coyote	ACME product du jour
2	(TEST) Scooby	dog	1969-05-19	2000-02-11	fake ghosts	mask
3	(TEST) Perry	platypus	2013-01-20	2015-04-09	H. Doofensmirtz	...inator
4	(TEST) Mr. Krabs	crab	2010-06-17	2014-07-07	Plankton	various
5	(TEST) Bugs Bunny	rabbit	1966-05-22	1988-04-15	E. Fudd	wabbit gun

END OF LAB