Docker Containers

Lab 4 – Power User Container Features

In this lab you will get a chance to do more advanced Docker container management.

1. Docker inspect

The `docker container inspect` subcommand provides access to all of the meta data associated with a container. To
experiment with inspect we'll create a container with several known bits of meta data, like a name and labels. Create
the following container:

```
user@ubuntu:~$ docker container create --name=websvr5 \
--hostname=ws05 --label="copyright=2017" --label="lic=apache2" httpd

Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
693502eb7dfb: Already exists
07ea63cb951e: Pull complete
e523938ce387: Pull complete
35123eac43ec: Pull complete
64b018d9be38: Pull complete
bd9836efbe75: Pull complete
ddb10b12e5d3: Pull complete
Digest: sha256:4612fba4347bd87eaeecd5c522d844f26cc4065b45eef9291277497946b7a86c
Status: Downloaded newer image for httpd:latest
c3aca7a917bc1a6527108889e3e0b372f4e700c35f43dd37ea830dd468719a5a

user@ubuntu:~$
```

Docker stores the metadata for containers in it operating directory under a subdirectory called "containers". Locate
Docker's operating directory:

```
user@ubuntu:~$ docker info |& grep -i "docker root"

Docker Root Dir: /var/lib/docker

user@ubuntu:~$
```

You can simply run `docker info` if you like but the above `grep` command selects just the output we need, the Docker
root directory. This directory offers only limited access to non root users. Become *root* and list the contents of the
Docker root, locate the container metadata directory and display its contents:

```
user@ubuntu:~$ ls -l /var/lib/docker

ls: cannot open directory '/var/lib/docker': Permission denied

user@ubuntu:~$ sudo su

root@ubuntu:/home/user# ls -l /var/lib/docker/

total 36
drwx------ 5 root root 4096 Mar  7 20:01 aufs
drwx------ 3 root root 4096 Mar  7 20:48 containers
drwx------ 3 root root 4096 Mar  7 20:01 image
drwxr-x--- 3 root root 4096 Mar  7 20:01 network
drwx------ 4 root root 4096 Mar  7 20:01 plugins
drwx------ 2 root root 4096 Mar  7 20:01 swarm
drwx------ 2 root root 4096 Mar  7 20:48 tmp
drwx------ 2 root root 4096 Mar  7 20:01 trust
drwx------ 2 root root 4096 Mar  7 20:01 volumes

root@ubuntu:/home/user# ls -l /var/lib/docker/containers/

total 4
drwx------ 3 root root 4096 Mar  7 20:48 c3aca7a917bc1a6527108889e3e0b372f4e700c35f

root@ubuntu:/home/user#
```

Docker creates a subdirectory for each container created under the containers directory. List the contents of the
containers directory:

```
root@ubuntu:/home/user# ls -l /var/lib/docker/containers/c3aca7a917bc1a6527108889e3

total 12
drwx------ 2 root root 4096 Mar  7 20:48 checkpoints
-rw-r--r-- 1 root root 2196 Mar  7 20:48 config.v2.json
-rw-r--r-- 1 root root 1117 Mar  7 20:48 hostconfig.json

root@ubuntu:/home/user#
```

This listing shows two files. One is the host independent configuration for our container (`config.v2.json`) and the
other is the host dependent configuration file (`hostconfig.json`). The host config contains things that might change
from host to host. Docker containers attempts to be as portable as possible so most of the interesting information is in
the `config.v2.json`.

Install `jq` to help review the files.

```
root@ubuntu:/home/user# apt-get install jq -y
...
```

List both files, first the `config.v2.json`:

```
root@ubuntu:/home/user# cat /var/lib/docker/containers/c3aca7a917bc1a6527108889e3e0

{
  "StreamConfig": {},
  "State": {
    "Running": false,
    "Paused": false,
    "Restarting": false,
    "OOMKilled": false,
    "RemovalInProgress": false,
    "Dead": false,
    "Pid": 0,
    "ExitCode": 0,
    "Error": "",
    "StartedAt": "0001-01-01T00:00:00Z",
    "FinishedAt": "0001-01-01T00:00:00Z",
    "Health": null
  },
  "ID": "c3aca7a917bc1a6527108889e3e0b372f4e700c35f43dd37ea830dd468719a5a",
  "Created": "2017-03-08T04:48:14.595321354Z",
  "Managed": false,
  "Path": "httpd-foreground",
  "Args": [],
...
```

now the `hostconfig.json`:

```
root@ubuntu:/home/user# cat /var/lib/docker/containers/c3aca7a917bc1a6527108889e3e0
```

```
{
  "Binds": null,
  "ContainerIDFile": "",
  "LogConfig": {
    "Type": "json-file",
    "Config": {}
  },
  "NetworkMode": "default",
  "PortBindings": {},
  "RestartPolicy": {
    "Name": "no",
    "MaximumRetryCount": 0
  },
  "AutoRemove": false,
  "VolumeDriver": "",
  "VolumesFrom": null,
  "CapAdd": null,
  "CapDrop": null,
  "Dns": [],
  "DnsOptions": [],
  "DnsSearch": [],
...
```

The file format is JSON, but without using `jq` it will appear to be a single line which is hard to review.

- Consider your copyright label from the container in question, which file do you think it is in? Why?
- Do you think you will find a process ID (PID) in the metadata? Why or why not?

Next lets start the container we created and see what happens in the container's directory, first list:

```
root@ubuntu:/home/user# docker container ls -a

CONTAINER ID        IMAGE     COMMAND              CREATED             STATUS
c3aca7a917bc        httpd     "httpd-foreground"   10 minutes ago      Created

root@ubuntu:/home/user#
```

Now start

```
root@ubuntu:/home/user# docker container start websvr5
```

```
websvr5

root@ubuntu:/home/user#
```

list again

```
root@ubuntu:/home/user# docker container ls

CONTAINER ID        IMAGE     COMMAND              CREATED             STATUS
c3aca7a917bc        httpd     "httpd-foreground"   11 minutes ago      Up 37 second

root@ubuntu:/home/user#
```

Now re-list the container directory:

```
root@ubuntu:/home/user# ls -l /var/lib/docker/containers/c3aca7a917bc1a6527108889e3

total 32
-rw-r----- 1 root root  893 Mar  7 20:59 c3aca7a917bc1a6527108889e3e0b372f4e700c35f
drwx------ 2 root root 4096 Mar  7 20:48 checkpoints
-rw-r--r-- 1 root root 3041 Mar  7 20:59 config.v2.json
-rw-r--r-- 1 root root 1117 Mar  7 20:59 hostconfig.json
-rw-r--r-- 1 root root    5 Mar  7 20:59 hostname
-rw-r--r-- 1 root root  166 Mar  7 20:59 hosts
-rw-r--r-- 1 root root  194 Mar  7 20:59 resolv.conf
-rw-r--r-- 1 root root   71 Mar  7 20:59 resolv.conf.hash
drwxrwxrwt 2 root root   40 Mar  7 20:59 shm

root@ubuntu:/home/user#
```

To start our container, Docker needs to create more support files. The `.log` file contains the log output of the
container, this allows us to recover the container's log data even after it has stopped. The `hostname`, `hosts`, and
`resolv.conf` are all standard networking files the Docker daemon will place in the container's `etc` directory. Note
that if we started this image (httpd) five times, each container, while otherwise identical, would need a unique
hostname, IP address and perhaps DNS information. These files take care of the filesystem side of these issues.

Docker can update the `resolv.conf` when DHCP changes the DNS settings for the host

network, the `resolv.conf.hash` allows Docker to detect changes to the `resolv.conf` file, avoiding modifications if the user has made changes of their own. The `shm` directory is used by the shared memory system, each container has its own isolated shared memory by default.

A lot has changed since we started our container. Redisplay the contents of your `config.v2.json` file:

```
root@ubuntu:/home/user# cat /var/lib/docker/containers/c3aca7a917bc1a6527108889e3e0

{
  "StreamConfig": {},
  "State": {
    "Running": true,
    "Paused": false,
    "Restarting": false,
    "OOMKilled": false,
    "RemovalInProgress": false,
    "Dead": false,
    "Pid": 5852,
    "ExitCode": 0,
    "Error": "",
    "StartedAt": "2017-03-08T04:59:09.73654752Z",
    "FinishedAt": "0001-01-01T00:00:00Z",
    "Health": null
  },
  "ID": "c3aca7a917bc1a6527108889e3e0b372f4e700c35f43dd37ea830dd468719a5a",
  "Created": "2017-03-08T04:48:14.595321354Z",
  "Managed": false,
  "Path": "httpd-foreground",
  "Args": [],
...
```

Examine the Pid value in the before and after files. As you can see, only running containers have process IDs.

```
root@ubuntu:/home/user# cat /var/lib/docker/containers/c3aca7a917bc1a6527108889e3e0

5852

root@ubuntu:/home/user#
```

While all of this under the hood exploration is interesting, and certainly informative, it is not very portable. A new
version of Docker could move all of these things around, store the metadata in a database, or what have you.

In production the best way to collect metadata on a container is to use the `container inspect` subcommand. Run `docker container inspect` on your container:

```
root@ubuntu:/home/user# docker container inspect websvr5 | more

[
    {
        "Id": "c3aca7a917bc1a6527108889e3e0b372f4e700c35f43dd37ea830dd468719a5a",
        "Created": "2017-03-08T04:48:14.595321354Z",
        "Path": "httpd-foreground",
        "Args": [],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 5852,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2017-03-08T04:59:09.73654752Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:f316d5949bb02561a68216997d2d7a3b80d1f621729d50e1f14a4172af
        "ResolvConfPath": "/var/lib/docker/containers/c3aca7a917bc1a6527108889e3e0b
...
```

Notice that the `inspect` output has a large section for `HostConfig` and another for `Config`. Via `jq` we can look at
each:

- `docker container inspect websvr5 | jq .[].HostConfig`
- `docker container inspect websvr5 | jq .[].Config`

Now we know where these come from. You can also see the `log` file, `resolv.conf`, `hostname`, and `hosts` file
references here. We'll look at these and the networking section further later in the course.

Use the `-f` filter switch as demonstrated in class to display the following container metadata from your web container:

- Pid (hint `docker container inspect websvr5 -f '{{.State.Pid}}'`)
- IPAddress
- Labels
- Environment variables
- Container name

2. CLI config.json

Using the text as a guide, create a `.docker/config.json` CLI configuration file that uses the *psFormat* key to
customize your default `docker container ls` output. Make your custom format display ID, labels, name, and command at a
minimum.

Test and debug your *ps* config.

Note: If you have not issued a Docker command requiring Docker to create the `config.json` it may not exist. You can
then create the `.docker` directory and the `config.json` yourself.

- Learn more here https://docs.docker.com/engine/reference/commandline/cli/

3. Exploring a Container

- Use `docker container inspect` to show the command (Path) used to launch your Docker web server container from earlier
  in the lab
- Exec a bash shell into the container locate this file (try: # which)
- `cat` the contents of the file
- What does the last line of the file do?
- Open a new terminal and run this command: `$ man exec`, read the first line of the description
- When the httpd-foreground script runs, what is the PID of the startup shell running the script?
- Why is the exec command so important?

4. Signals

In this step we will explore Docker and signals. Run an Ubuntu container that executes the `sleep 30` command:

```
root@ubuntu:/home/user# docker container run -itd ubuntu sleep 30

794b4237dffab04730650fe8f729c805e53dc9da346bdf25735f5bdbfd8f720a

root@ubuntu:/home/user#
```

This container will sleep for 30 seconds and then exit.

Now run `docker container wait` to wait for the Docker container sleeping to exit:

```
user@ubuntu:~$ docker container wait 794b; echo "its done"

0
its done
```

When the first container exits the `container wait` subcommand returns allowing the `echo` command to run. This can be a
useful tool in many contexts. Imagine you are creating a script and the commands executing in the script need to be run
serially.

Containers are typically run asynchronously but you could force the script to wait for the container's completion using
the `container wait` subcommand.

You can use the `container start` or `container restart` subcommand to restart the original container after it exits.
Try it:

```
root@ubuntu:/home/user# docker container restart 794b

794b

root@ubuntu:/home/user#
```

Imagine you have a flakey service that keeps crashing after running for 5 hours. The team is working on a fix but you
need to keep the service running in the mean time. In most cases you would want to run a fresh copy of the container
from the image, however if the container can be safely restarted, you can give Docker a restart

policy.

Rerun you sleep container with a restart policy of *always*:

```
root@ubuntu:/home/user# docker container run -itd --restart=always ubuntu sleep 30

17869ad8397fd57abac7e2e6e5680b6243bc5c82e955b0734baf9307e4ac630a

root@ubuntu:/home/user# docker container ls -f Id=1786

CONTAINER ID        IMAGE      COMMAND        CREATED             STATUS
17869ad8397f        ubuntu     "sleep 30"     11 seconds ago      Up 10 seconds

root@ubuntu:/home/user# sleep 45

root@ubuntu:/home/user# docker container ls -f Id=1786

CONTAINER ID        IMAGE      COMMAND         CREATED              STATUS
17869ad8397f        ubuntu     "sleep 30"      About a minute ago   Up 5 seconds

root@ubuntu:/home/user#
```

In the example above, even when we wait 45 seconds we find the sleep container running. Docker will always restart it if
it terminates.

5. Events

Events can help us better understand the happenings within a given Docker engine. For example, in the example above it
is difficult to detect the application failure (sleep exiting). We can see the run times are short in the `container ls`
listings but there is a better way to track application `start` and `stop` events. Run the `docker events` subcommand:

```
root@ubuntu:/home/user# docker events

2017-03-07T21:16:37.757638263-08:00 container die 17869ad8397fd57abac7e2e6e5680b624
2017-03-07T21:16:37.834595485-08:00 network disconnect e943915b1cb25ede9671dfc0a05e
2017-03-07T21:16:37.870784766-08:00 network connect e943915b1cb25ede9671dfc0a05e352
2017-03-07T21:16:38.073727100-08:00 container start 17869ad8397fd57abac7e2e6e5680b6
^C
```

Within 30 seconds you should see your sleep container exit (when the sleep expires) and then get started by Docker due
to the restart policy.

6. Cleanup

After you have finished exploring, stop, and remove all of the containers you started/created in this lab. Remember that
you can refer to containers by name or ID.

Congratulations you have completed the advanced controlling containers lab!