RX-M

Docker Containers

Lab 3 – Controlling Containers

In this lab you will get a chance to manage Docker containers with a range of Docker CLI features.

1. Run container as a service

When processes are run in the background they are often referred to as daemons. You can run containers in the background as daemons with the `docker run -d` switch. The *-d* switch stands for "detached" and runs the container detached from your shell's input and output.

Imagine we have a service and that its job is to log data to STDOUT every 10 seconds. We can run this service in the background with the `docker run -d` switch. Execute the following Docker command:

```
user@ubuntu:~$ docker run -d ubuntu /bin/sh -c "while true; do echo 'hello docker'; sleep 10; done"
7e31017163eab29eecc2e83fab9f74fea68f6da9ffb23f92b0ce278dc9df8a4b
```

When the container starts, display the running containers. You should see the new container running in the background.

The `docker logs` subcommand can display the output of a background container. Display the STDOUT log for the container you started above.

```
user@ubuntu:~$ docker logs 7e31
hello docker
hello docker
hello docker
```

- Get help on the `docker logs` subcommand
- Display the container log data with timestamps added
- Use the "follow" option to continuously display the log output of the container

Next let's run an actual service within a container. Nginx is a popular web server with an available image on Docker Hub. Run an Nginx container instance:

```
user@ubuntu:~$ docker run -d nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
efd26ecc9548: Pull complete
a3ed95caeb02: Pull complete
83f52fbfa5f8: Pull complete
fa664caa1402: Pull complete
Digest: sha256:12127e07a75bda1022fbd4ea231f5527a1899aad4679e3940482db3b57383b1d
Status: Downloaded newer image for nginx:latest
9c0539b42b4cdd44a3d5b7ab3c969b218d4de165279e44015ecab6e0c7fdc245
user@ubuntu:~$
```

Use the `docker top` subcommand to display the processes running within the container:

```
user@ubuntu:~$ docker top 9c05
UID      PID    PPID   C   STIME   TTY   TIME       CMD
Root     3913   3889   0   15:27   ?     00:00:00   nginx: master process nginx -g daemon off;
lightdm  3928   3913   0   15:27   ?     00:00:00   nginx: worker process
user@ubuntu:~$
```

Imagine you are a BSD Unix fan and prefer an "aux" style ps output to the default top output. You can have top display any ps style you like by simply including the *ps* options after the container ID/Name. For example:

```
user@ubuntu:~$ docker top 9c05 aux
USER     PID    %CPU   %MEM   VSZ     RSS    TTY   STAT   START   TIME   COMMAND
root     7137   0.0    0.2    31688   5024   ?     Ss     15:06   0:00   nginx: master process nginx -g
daemon off;
syslog   7167   0.0    0.1    32112   2900   ?     S      15:06   0:00   nginx: worker process
```

- Run the top command with the -t ps switch
- Run the top command with the -w ps switch
- Try your own favorite ps switches with `docker top`

The concept of a “container” is implemented via kernel namespaces and cgroups. Many processes can run within the isolation/constraints of the same “container”. Notice that this container has no shell running within it. This is typical of service/microservice containers. You can still launch a shell within the container to perform diagnostics using the `docker exec` subcommand.

Run a new interactive shell within the Nginx container:

```
user@ubuntu:~$ docker exec -it 9c05 /bin/sh
#
```

Now list all of the processes running within the container:

```
# ps -ef
UID        PID  PPID  C STIME TTY TIME     CMD
root         1     0  0 22:27 ?   00:00:00 nginx: master process nginx -g daemon off;
nginx        5     1  0 22:27 ?   00:00:00 nginx: worker process
root         6     0  0 22:33 ?   00:00:00 /bin/sh
root        11     6  0 22:34 ?   00:00:00 ps -ef
#
```

In this listing we can see the two processes launched with the container and the shell we have launched within the container along with the ps command running under the shell.

Also note that the “top” command run from the host shows the host relative process ids (the main nginx process id was 3913 in the above example.) Within the container the main nginx process has process id 1. This is the manifestation of the container’s process namespace. Processes within a container have one id in the container and another id on the host.

The first process launched within each container is always process id 1 within the container.

Display the IP addresses within the container:

```
# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
20: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:06 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.6/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:6/64 scope link
       valid_lft forever preferred_lft forever
```

Now exit the container and see if you can request the root page from the nginx service from the host with `wget`:

```
# exit

user@ubuntu:~$ wget -qO - 172.17.0.6 | head
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
```

2. Creating and copying from containers

In this step we're going to use the BusyBox container image. BusyBox combines tiny versions of many common UNIX utilities into a single small executable. It provides replacements for most of the utilities you usually find in GNU fileutils, shellutils, etc. The utilities in BusyBox generally have fewer options than their full-featured GNU cousins; however, the options that are included provide the expected functionality and behave very much like their GNU counterparts. BusyBox provides a fairly complete environment for any small or embedded system.

Due to its small size and fairly complete feature set, BusyBox makes for a good test container image.

Imagine you need the `uuencode` tool to encode some strings for the web. Unfortunately Ubuntu 16.04 doesn't come with uuencode. To avoid installing a big Ubuntu package we could just grab a copy of the small uuencode tool from busybox. You don't need to run a BusyBox container to copy files out of the image. We can simply create a container based on busybox and `docker cp` the files we need. Try it:

```
user@ubuntu:~$ docker create -it --name file-donor busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
8ddc19f16526: Pull complete
Digest: sha256:a59906e33509d14c036c8678d687bd4eec81ed7c4b8ce907b888c607f6a1e0e6
Status: Downloaded newer image for busybox:latest
db7f0716bea9964e8dc4081240cc40572295dad605c03571530dd1630ec54401
```

Now copy out the binary you need and use it to encode your string:

```
user@ubuntu:~$ which uuencode   # No uuencode :-(
user@ubuntu:~$ docker cp file-donor:/bin/uuencode ./uuencode
user@ubuntu:~$ ls -l uuencode
```

```
-rwxr-xr-x 1 user user 1010960 Jun 23 13:13 uuencode
user@ubuntu:~$ echo "diphthong-test: How now brown cow?" | ./uuencode -
begin 664 -
C9&EP:'1H;VYG+71E<W0Z($AO=R!N;W<@8G)O=VX@8V]W/PH`
`
end
user@ubuntu:~$
```

Perfect!

3. Starting and stopping containers

We can stop and restart containers using the `docker stop` and `docker start` subcommands. Let's try this by stopping and restarting the file-donor container created (but not run) above. First start the file-donor container:

```
user@ubuntu:~$ docker start -i file-donor
/ #
user@ubuntu:~$
```

Explore busybox for a minute if you like and then detach from the shell with `^p ^q`.

To stop a container, use `docker stop` and pass the command the ID of the container (you can use the entire ID or just a prefix, as long as you supply enough characters to make the prefix unique on the local Docker host.) For example, use `docker ps` subcommand to display your file-donor container and then use the `docker stop` subcommand to stop it:

```
user@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
daa171067ccc        busybox             "sh"                     3 minutes ago       Up About a minute
file-donor

user@ubuntu:~$ docker stop file-donor
file-donor

user@ubuntu:~$
```

N.B. your container ID will not be the same as the one in this example.

Run the `docker ps` subcommand again.

- How many containers are running?

Containers are not destroyed unless you explicitly remove them. You can delete a container with the `docker rm` command. Run the `docker ps` subcommand with the "-a" all switch.

```
user@ubuntu:~$ docker ps -a
```

- How many total containers are present on your Docker host?
- How many containers are stopped on your Docker host?

Rerun the file-donor container again using the `docker start` subcommand, something like this:

```
user@ubuntu:~$ docker start -i file-donor
/ #
```

The `start` subcommand reruns the existing container, re-executing the container's command ("/bin/bash" in this case.) The "-i" switch connects the input stream of you shell to the container.

4. Running command containers

In class you discussed how containers can be used as services, machines, and commands. Think back to your use of uuencode earlier in the lab. Rather than copying the uuencode program to our host and running it, we could have just ran it in the busybox container. In fact this would not only be easier but more reliable. After all, our host could have had some incompatible libraries installed or missing configuration, etc.

Try running the uuencode program directly within the busybox container:

```
user@ubuntu:~$ docker run busybox /bin/uuencode
BusyBox v1.25.0 (2016-06-23 20:12:29 UTC) multi-call binary.

Usage: uuencode [-m] [FILE] STORED_FILENAME

Uuencode FILE (or stdin) to stdout

    -m    Use base64 encoding per RFC1521
```

Ok, so far so good. We now know we can run any program in the container filesystem within the container. The help tells us that the uuencode program will accept input from stdin. Most Unix programs allow you to send data to them via stdin either by default or by passing them the dash ("-") argument.

So if we echo our encoding string to the "uuencode -" we should get our encoded string. The only catch is that we will not be running the uuencode program directly, we will be running Docker. We need to tell Docker to connect the stdin of uuencode to our shell. This is easy enough, we have been using the "-i" switch for this. However, in this case we do not want an interactive tty session (-t), we simply want to connect our output to the uuencode program in the container, letting its output flow back to our terminal.

Try it:

```
user@ubuntu:~$ echo "diphthong-test: How now brown cow?" | docker run -i busybox /bin/uuencode -
begin 644 -
C9&EP:'1H;VYG+71E<W0Z($AO=R!N;W<@8G)O=VX@8V]W/PH`
`
end
user@ubuntu:~$
```

Excellent. Now we have seen containers in the role of machines (our Ubuntu containers), in the role of services (our nginx container) and commands (our uuencode busybox.)

5. Stats

Run a new nginx container as a daemon:

```
user@ubuntu:~$ docker run -d --name=web87 --label="svc=web" nginx
e9fd14834bf46e138e2f8f2f8bfb3a4b52d5b72af42a803180c440a3ec0c0329
```

Run the `docker stats` subcommand.

- How many containers are running?
- How many process IDs are present in each container?
- Other than the container ID, what type of data is stats showing you?

Type `control C` to exit stats.

Rerun `docker stats` with the *web87* argument:

```
user@ubuntu:~$ docker stats web87
```

- What is displayed?

Stop stats and rerun stats with the -a switch:

```
user@ubuntu:~$ docker stats -a
```

- What is displayed?

Exit stats.

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 Controlling Containers lab!