RX-M

Docker Containers

Lab 2 – Running Containers

In this lab you will get a chance to start working with Docker containers. In the steps ahead you will create a number of containers to become familiar with the common tasks involved in using Docker.

As you start multiple containers with interactive prompts you will need to take care to keep track of which shells in your lab VM are interacting with Containers and which are interacting with the Host.

- Host shell prompts will look like this: **user@ubuntu:~$**
- Container shell prompts will look something like this: **root@936793e216e3:/#**

The container ID is used as the container hostname and appears in the bash prompt of most container images by default. Remember that Docker is installed on the host. Due to container isolation, containers do not have access to the host and therefore you cannot invoke the Docker command from within a container (unless you install Docker in the container!) Anytime the lab asks you to run a `docker` command, you will need to do this from the host prompt, not from within a container.

1. Run an Ubuntu machine container

As a first container experiment you will run an instance of the Ubuntu image with the "latest" tag from the Docker Hub registry. You will launch a bash shell interactively in the new container so that you can explore various aspects of the container. To do this you will need to use the following `docker run` arguments:

- **-t** Allocates a sudo tty in the container
- **-i** Connects the STDIN of your host shell to the container tty
- <**repository**> The repository where the container image will be found, "ubuntu" in this case
- <**cmd**> The command to run within the container, "/bin/bash" in this case

Examine the command line help (`docker help run`) and the online reference (https://docs.docker.com/reference/commandline/run) for the `docker run` command. The run subcommand is the most feature rich Docker command, supporting a wide array of command line arguments. It is also perhaps the most important command, as it is the command which creates running containers. We will start simple and then add more complex options to our run commands as the class develops.

Your run command should look something like this:

```
user@ubuntu:~$ docker run -it ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
2f0243478e1f: Pull complete
d8909ae88469: Pull complete
820f09abed29: Pull complete
01193a8f3d88: Pull complete
Digest: sha256:8e2324f2288c26e1393b63e680ee7844202391414dbd48497e9a4fd997cd3cbf
Status: Downloaded newer image for ubuntu:latest
root@dce36fe53d55:/#
```

If you are running this command for the first time, Docker will not be able to find the Ubuntu image locally and will automatically download it from Docker Hub. Once you have the Ubuntu container running, display the contents of the

`/etc/os-release` file from within the container:

```
root@373262a6f3f7:/# cat /etc/os-release
...
```

- What Distribution and version of Linux is configured in the container?
- What user are you inside the container?
- What is the ID of the container?

Next use the `uname` command to discover the kernel version.

```
root@373262a6f3f7:/# uname -a
...
```

- What version of the Linux Kernel is running?
- What organization built it?
- What is the hostname?

Open a second terminal in the Lab VM and run the above two commands from outside of the container.

- What Distribution and version of Linux is configured on the host?
- What user are you on the host?
- What Linux Kernel version does the host report?
- What is the hostname?

2. Run a CentOS machine container

In a new terminal, run a second container just like the first but using the “centos” repository latest image and adding a name for the container:

```
user@ubuntu:~$ docker run -it --name cen7 centos /bin/bash
Unable to find image 'centos:latest' locally
latest: Pulling from library/centos

a3ed95caeb02: Pull complete
5989106db7fb: Pull complete
Digest: sha256:1b9adf413b3ab95ce430c2039954bb0db0c8e2672c48182f2c5b3d30373d5b71
Status: Downloaded newer image for centos:latest
[root@269696a7e988 /]#
```

Use the commands from the Ubuntu example, answer these questions:

- What Distribution and version of Linux is configured inside the container?
- What version of the Linux Kernel does the container report?
- What is the hostname?
- Is the hostname the same as the container name?

3. Run an Ubuntu 12.04 machine container

Every container is based on precisely one image. All images have an ID. Images are grouped into named repositories. Images within a repository are given “tag” names. When running a container the image the container will be based on must be specified. You can reference an image by ID (e.g. f1b10cd84249) or repository:tag name (e.g. Ubuntu:14.04.) You can also refer to an image by repository name alone (e.g. Ubuntu). When no tag is specified the “latest” tag is implied. The

Ubuntu repository offers a 12.04 tag to run the older LTS version of Ubuntu. Launch a 12.04 container in a new host terminal by specifying an explicit tag name and give it the hostname u12.

```
user@ubuntu:~$ docker run -it -h u12 ubuntu:12.04
Unable to find image 'ubuntu:12.04' locally
12.04: Pulling from library/ubuntu

765826873799: Pull complete
e7a187926114: Pull complete
fd01d4f3de3b: Pull complete
c704fce22a3c: Pull complete
Digest: sha256:45bf6eb4403c7171dc497a7105c2ef3acd12f16b4b80256b067fc2a8183c6348
Status: Downloaded newer image for ubuntu:12.04
root@u12:/#
```

- What distribution and version of Linux is configured inside the container?
- What version of the Linux kernel does the container report?
- How many actual kernels are running?
- How many processes do you think are running inside the container?
- What is your user id inside the container?
- What is the hostname inside the container?

Display a full process list inside the container and list your user id:

```
root@u12:/# ps -ef
UID         PID   PPID  C STIME TTY          TIME CMD
root          1      0  0 21:31 ?        00:00:00 /bin/bash
root         10      1  0 21:32 ?        00:00:00 ps -ef

root@u12:/# id
uid=0(root) gid=0(root) groups=0(root)
```

- How many processes are running on the host system?
- What user are you logged in as on the host?
- What shell are you using in the container?
- Did you ask for that shell in the `docker run` command line?

Container images almost always have default commands which run when no executable is specified with the `docker run` command.

Display the IP addresses provided inside the container:

```
root@u12:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    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
16: eth0@if17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.4/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:4/64 scope link
       valid_lft forever preferred_lft forever
root@u12:/#
```

- What IP addresses are available inside your container?
- What IP addresses are available on the host system?
- From a shell on the host try to ping the IP address of the container, does it work?

4. Listing containers

In a new terminal on the host run the `docker ps` subcommand to view all of the running containers.

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

- How many containers are running?
- What is the ID of the first container in the ps list?
- Is this the complete ID of that container?

Run the following command from the host prompt:

```
user@ubuntu:~$ docker ps --no-trunc
```

- How is the output different from the prior command?
- What is the name of the first container in the ps list?
- What command is the first container in the ps list running?
- Are all of the containers listed running?

Docker does not automatically delete containers whether they terminate normally or abnormally. To see running and terminated containers you can use the `docker ps -a` switch. Try it:

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

You should see the hello-world container added to the list of containers.

To get help with any Docker command you can enter the Docker command followed by --help. Get help on the *ps* subcommand:

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

Usage:    docker ps [OPTIONS]

List containers

Options:
  -a, --all             Show all containers (default shows just running)
  -f, --filter value    Filter output based on conditions provided (default [])
      --format string   Pretty-print containers using a Go template
      --help            Print usage
  -n, --last int        Show n last created containers (includes all states) (default -1)
  -l, --latest          Show the latest created container (includes all states)
      --no-trunc        Don't truncate output
  -q, --quiet           Only display numeric IDs
  -s, --size            Display total file sizes
```

Using the help and the course book, perform the following container listing with `docker ps`:

- List all of the containers created since your ubuntu:12.04 container
- List all of the containers created before your ubuntu:12.04 container

- List all of the containers that have exited with error code 0 (hint: this only works with the -a switch)
- List all of the containers based on the ubuntu repository
- Rerun all of the above commands displaying only the id, name and command of the container

5. Using labels

Labels are key/value meta data associated with containers. Docker stores labels in the container configuration but pays no attention to them. In this way, external programs can use labels to tag containers (and other Docker artifacts as we will see) for their own organizational purposes. If you have some containers that are part of the test system and some that are part of the production system running on the same host, you could give some of the containers the "test" label and some the "production" label.

Labels can have just a key, or a key and a value. Create a new container with the label: *env=test*

```
user@ubuntu:~$ docker run -itd --label="env=test" ubuntu:12.04
d3830451c6cc4711235165a455d4ec4f5bd18fad91f1d844cd56dfd9b523177f
```

Now use the `docker ps` subcommand to display only containers with the desired label: *env=test*

```
user@ubuntu:~$ docker ps -f label="env=test"
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
PORTS               NAMES
d3830451c6cc        ubuntu:12.04        "/bin/bash"         23 seconds ago      Up 22 seconds
adoring_murdock
user@ubuntu:~$
```

- Create a second background container with the label "env=production"
- Run a `docker ps` subcommand that only displays the env production value
- Run a Docker command that displays any container with an env value of any type

6. Removing containers

Stopped containers will stay on your system perpetually unless deleted. You can delete unneeded containers with the `docker rm` subcommand. You can also run interactive containers with the “--rm” switch, which will cause the container to self-delete when you exit the container.

Execute Docker commands to perform the following:

- Display all (-a) of the containers on your system
- Use the `docker rm` subcommand to remove one of the exited containers by ID
- Try to use `docker rm` to remove a running container, what happens?
- Next try to remove the same container adding the --force switch, what happens?
- Run the `docker ps -aq` subcommand, what happens?
- Finally, to quickly clean up your Docker lab system, run the `docker rm --force` subcommand on the output of a subshell executing the `docker ps -aq` subcommand:

```
user@ubuntu:~$ docker rm --force `docker ps -aq`
d7dd23f3f9ac
5edca5862986
6258564cf349
dce36fe53d55
081fcf274729
user@ubuntu:~$
```

Congratulations you have completed the Running Containers lab!