Docker Containers

Docker containers support Operating-system-level virtualization. Containerizing applications with docker involves the
use of operating system features in which the kernel allows the existence of multiple isolated user-space instances.
Such instances are called containers, partitions, virtualization engines (VEs) or jails (FreeBSD jail or chroot jail) in
different OS environments. Containers may look like real computers from the point of view of programs running in them.

A computer program running on an ordinary computer's operating system can see all resources (connected devices,
files and folders, network shares, CPU power, quantifiable hardware capabilities) of that computer. However, programs
running inside a container can only see the container's contents and devices assigned to the container. On Unix-like
operating systems, this feature can be seen as an advanced implementation of the standard chroot mechanism, which
changes the apparent root folder for the current running process and its children. In addition to isolation mechanisms,
the kernel often provides resource-management features to limit the impact of one container's activities on other
containers.

Lab 2 – Running containers

In this lab you will get a chance to focus on controlling Docker containers. In the steps ahead you will create and
control 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 container run` arguments:

`docker container run -t -i ubuntu /bin/bash`

- -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 container run --help`) and the online reference (https://docs.docker.com/engine/reference/commandline/container_run/) for the `docker container run` command. The `run`
subcommand is the most feature rich Docker command, supporting a wide array of command line arguments and switches. 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.

Use the command above to launch an Ubuntu container:

```
user@ubuntu:~$ docker container run -t -i ubuntu /bin/bash

Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
d54efb8db41d: Pull complete
f8b845f45a87: Pull complete
e8db7bf7c39f: Pull complete
```

```
9654c40e9079: Pull complete
6d9ef359eaaa: Pull complete
Digest: sha256:dd7808d8792c9841d0b460122f1acf0a2dd1f56404f8d1e56298048885e45535
Status: Downloaded newer image for ubuntu:latest

root@6ec4354a2dc0:/#
```

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@6ec4354a2dc0:/# cat /etc/os-release

NAME="Ubuntu"
VERSION="16.04.2 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.2 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
VERSION_CODENAME=xenial
UBUNTU_CODENAME=xenial

root@6ec4354a2dc0:/#
```

- 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@6ec4354a2dc0:/# uname -a

Linux 28081230d5ad 4.4.0-89-generic #112-Ubuntu SMP Mon Jul 31 19:38:41 UTC 2017 x8
root@6ec4354a2dc0:/#
```

- What version of the Linux Kernel is running?
- Which 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 container run -it --name cent7 centos /bin/bash

Unable to find image 'centos:latest' locally
latest: Pulling from library/centos
45a2e645736c: Pull complete
Digest: sha256:c577af3197aacedf79c5a204cd7f493c8e07ffbce7f88f7600bf19c688c38799
Status: Downloaded newer image for centos:latest

[root@01e914a9db5e /]#
```

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

- Which 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 terminal
and specify an explicit hostname "u12".

```
user@ubuntu:~$ docker container run -it -h u12 ubuntu:12.04

Unable to find image 'ubuntu:12.04' locally
12.04: Pulling from library/ubuntu
7d172a1d710c: Pull complete
0d35f4bc65ee: Pull complete
19231e74a42a: Pull complete
51880de525fb: Pull complete
46b3aaaab3ec: Pull complete
Digest: sha256:47effe29d7bed549fee8c63bd60120fa88785e72abfb2aeeccdda21f4eae67e4
Status: Downloaded newer image for ubuntu:12.04

root@u12:/#
```

- Which 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?
- What is the "container name" of the container?
- Are the hostname and container name always the same in a 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 04:14 ?        00:00:00 /bin/bash
root          11      1  0 04:14 ?        00:00:00 ps -ef

root@u12:/# id

uid=0(root) gid=0(root) groups=0(root)

root@u12:/#
```

- 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 container run` command line?

Container images almost always have default commands which run when no executable is specified with the `docker container 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
10: eth0@if11: <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 container ls` subcommand to view all of the running containers.

```
user@ubuntu:~$ docker container ls
```

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

Run the following command from the host prompt:

```
user@ubuntu:~$ docker container ls --no-trunc
```

- How is the output different from the prior command?
- What is the name of the first container in the `container ls` list?
- What command is the first container in the `container ls` 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 container ls -a` switch.

Try it:

```
user@ubuntu:~$ docker container ls -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 `container ls` subcommand:

```
user@ubuntu:~$ docker container ls --help

Usage:    docker container ls [OPTIONS]

List containers

Aliases:
  ls, ps, list

Options:
  -a, --all             Show all containers (default shows just running)
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print containers using a Go template
      --help            Print usage
  -n, --last int        Show n last created containers (includes all states) (defau
  -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

user@ubuntu:~$
```

Using the Docker help and the course presentation, perform the following container listings with `docker container ls`:

- 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
- Using `ps -ef` or `pstree` show how containers are launched by Docker Engine.

Ex.

```
user@ubuntu:~$ pstree -s $(docker container top \
$(docker container ls -q | head -1) | grep bash | awk '{print $2}')

systemd───dockerd───docker-containe───docker-containe───bash

user@ubuntu:~$
```

and

```
user@ubuntu:~$ pstree -s $(ps -ef | grep "dockerd" | head -1 | awk '{print $2}')

systemd───dockerd─┬─docker-containe─┬─2*[docker-containe─┬─bash]
                  │                 │                    └─8*[{docker-containe}]]
                  │                 ├─docker-containe─┬─bash
                  │                 │                 └─7*[{docker-containe}]
                  │                 └─11*[{docker-containe}]
                  └─12*[{dockerd}]

user@ubuntu:~$
```

Try to relate your other containers.

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 container run -itd --label="env=test" ubuntu:12.04

8ea71643a606a15cf5664cce00f89e0e1b18a18a210c54612788f48bdbee8360

user@ubuntu:~$
```

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

```
user@ubuntu:~$ docker container ls -f label="env=test"

CONTAINER ID     IMAGE            COMMAND          CREATED             STATUS
8ea71643a606     ubuntu:12.04     "/bin/bash"      29 seconds ago      Up 29 seconds

user@ubuntu:~$
```

Now list containers with the same source image.

```
user@ubuntu:~$ docker container ls -f ancestor=ubuntu:12.04

CONTAINER ID      IMAGE            COMMAND          CREATED             STATUS
8ea71643a606      ubuntu:12.04     "/bin/bash"      52 seconds ago      Up 51 seconds
db637824280c      ubuntu:12.04     "/bin/bash"      13 minutes ago      Up 13 minutes

user@ubuntu:~$
```

- Create a second background container with the label "env=production"
- Run a `docker container ls` 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 container 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
- Stop the container via `docker container stop <cid>`
- Remove the container via `docker container rm <cid>`
- What is the `--force` switch used for on `rm`, what happens?
- One way to remove all running/stopped containers in one go looks like this:

```
user@ubuntu:~$ docker container rm $(docker container \
stop $(docker container ls -qa))

8ea71643a606
db637824280c
01e914a9db5e
6ec4354a2dc0
b813a517607f

user@ubuntu:~$ docker container ls -a

CONTAINER ID     IMAGE       COMMAND         CREATED         STATUS         PORTS

user@ubuntu:~$
```

Congratulations you have completed the running containers lab!