Hands-on With Docker

Docker: Provides isolation, multi-cloud platform usage, compatibility, and maintainability.

A docker image is an environment in the cloud, which you can download on your machine.

When you run the command docker pull ubuntu, you get a copy of ubuntu which is a Debian-based Linux operating system on your machine.

Using “docker pull ubuntu” you get an image with the name ubuntu, which has the basic ubuntu installed in it.
If you see the second line in the above image you can see that the tag given to this image is the latest, you can specify any other tag of your choice just to differentiate among multiple ubuntu images.
Now you locally have an ubuntu machine.

```
[node1] (local) root@192.168.0.28 ~
$ docker --version
Docker version 20.10.17, build 100c701
[node1] (local) root@192.168.0.28 ~
$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
dbf6a9befcde: Pull complete
Digest: sha256:dfd64a3b4296d8c9b62aa3309984f8620b98d87e47492599ee20739e8eb54fbf
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
[node1] (local) root@192.168.0.28 ~
$
```

$docker pull ubuntu

```
[node1] (local) root@192.168.0.28 ~
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
ubuntu       latest    3b418d7b466a   9 days ago   77.8MB
[node1] (local) root@192.168.0.28 ~
$
```

When you run the command ”docker images” you can find all the docker images present on your system.
Currently, you have only one image called ubuntu with the tag latest and an image ID and the time when this image was created on the docker website and also the size of this image.
You cannot view this as an image on any of the image viewers on your machine.

$docker images

Now, you have a docker image.

```
[node1] (local) root@192.168.0.28 ~
$ docker run -t -d --name latest ubuntu
5c7cb81d56ad70bc9c3674633f603094170f82aa1bc7beac83c9f72ad0115678
[node1] (local) root@192.168.0.28 ~
$
```

To run a docker image we use the command docker run. This will generate an ID.

$docker run -t -d --name latest ubuntu

When you run a docker image, you call it a container

When you run a docker image using the docker run command the process is called containerisation

and you have a copy of your docker image which is activated.

```
[node1] (local) root@192.168.0.28 ~
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS              PORTS     NAMES
5c7cb81d56ad   ubuntu    "/bin/bash"   About a minute ago   Up About a minute             latest
[node1] (local) root@192.168.0.28 ~
```

In the above image, you see a 12-digit container ID which is like the output of the image above this. It is your container ID. The name of the docker image used to run the container is ubuntu. The name of the container is the latest.

$docker ps -a

```
[node1] (local) root@192.168.0.28 ~
$ docker exec -it latest bash
root@5c7cb81d56ad:/#
```

There you go. Just type the above command. “docker exec -it latest bash” and you have a machine with the name root@5c7cb81d56ad. All the Linux commands work here.

$docker exec -it latest bash

```
[node1] (local) root@192.168.0.28 ~
$ docker exec -it latest bash
root@5c7cb81d56ad:/# ^C
root@5c7cb81d56ad:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
root@5c7cb81d56ad:/#
```

In the above image, I used the command “ls” to list what was there in my container root@5c7cb81d56ad.

```
root@5c7cb81d56ad:/# mkdir sherlin_samp
root@5c7cb81d56ad:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   sherlin_samp  sys  usr
boot  etc  lib   lib64  media   opt  root  sbin  srv           tmp  var
root@5c7cb81d56ad:/# cd sherlin_samp
root@5c7cb81d56ad:/sherlin_samp# touch my_text_file.txt
root@5c7cb81d56ad:/sherlin_samp# ls
my_text_file.txt
root@5c7cb81d56ad:/sherlin_samp#
```

The above image shows how my container works just like yet another system in my local machine.

How do you stop and delete a container?

```
[node1] (local) root@192.168.0.28 ~
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
5c7cb81d56ad   ubuntu    "/bin/bash"   13 minutes ago   Up 13 minutes             latest
[node1] (local) root@192.168.0.28 ~
```

docker ps -a will list all the running containers. Here you can see our running container.

Let us stop this container.

```
$ docker stop 5c7cb81d56ad
5c7cb81d56ad
[node1] (local) root@192.168.0.28 ~
```

After checking the running containers, you can choose to stop the one that you want. When you run this command container ID of the stopped container gets printed on the cmd window.

$docker stop 5c7cb81d56ad

```
[node1] (local) root@192.168.0.28 ~
$ docker stop 5c7cb81d56ad
5c7cb81d56ad
[node1] (local) root@192.168.0.28 ~
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS                       PORTS     NAMES
5c7cb81d56ad   ubuntu    "/bin/bash"   15 minutes ago   Exited (137) 28 seconds ago            latest
[node1] (local) root@192.168.0.28 ~
$
```

In the above image under the header STATUS, you see exited (137) 28 seconds ago. Because we stopped the container. Stop the container does not delete the container. To remove the container we must delete it.

```
[node1] (local) root@192.168.0.28 ~
$ docker rm 5c7cb81d56ad
5c7cb81d56ad
[node1] (local) root@192.168.0.28 ~
```

The command to delete a container is docker rm [CONTAINER ID]. Now your container is deleted.

$ docker rm [Container ID]

```
[node1] (local) root@192.168.0.28 ~
$ docker rm 5c7cb81d56ad
5c7cb81d56ad
[node1] (local) root@192.168.0.28 ~
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[node1] (local) root@192.168.0.28 ~
```

when you run the command “docker ps -a“ again. You do not see any container.

How to delete a docker image?

There is no process called as stopping the image before deleting it, unlike container. To delete an image the command is “docker rmi [Image name]”

```
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[node1] (local) root@192.168.0.28 ~
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
ubuntu       latest    3b418d7b466a   9 days ago   77.8MB
[node1] (local) root@192.168.0.28 ~
```

Run the command “docker images”. We have only one image which is ubuntu. Let us delete it.

```
$ docker images -a
REPOSITORY   TAG       IMAGE ID       CREATED      SIZE
ubuntu       latest    3b418d7b466a   9 days ago   77.8MB
[node1] (local) root@192.168.0.28 ~
$ docker rmi ubuntu
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:dfd64a3b4296d8c9b62aa3309984f8620b98d87e47492599ee20739e8eb54fbf
Deleted: sha256:3b418d7b466ac6275a6bfcb0c86fbe4422ff6ea0af444a294f82d3bf5173ce74
Deleted: sha256:b8a36d10656ac19ddb96ef3107f76820663717708fc37ce929925c36d1b1d157
[node1] (local) root@192.168.0.28 ~
$
```

The ubuntu image is deleted.

How to create your first Docker image and dockerize a Python Application

Dockerfiles enable you to create your own images. A Dockerfile describes the software that makes up an image. Dockerfiles contain a set of instructions that specify what environment to use and which commands to run.

Create a Dockerfile

```
[node1] (local) root@192.168.0.28 ~
$ mkdir first_python_docker
[node1] (local) root@192.168.0.28 ~
$ ls
first python docker
[node1] (local) root@192.168.0.28 ~
$ cd first_python_docker
[node1] (local) root@192.168.0.28 ~/first python docker
$ touch Dockerfile
[node1] (local) root@192.168.0.28 ~/first python docker
$ ls
Dockerfile
[node1] (local) root@192.168.0.28 ~/first python docker
$
```

Execute all the commands step by step to create a Dockerfile. You can create a file with name “Dockerfile” without any extension in the folder first_python_docker.

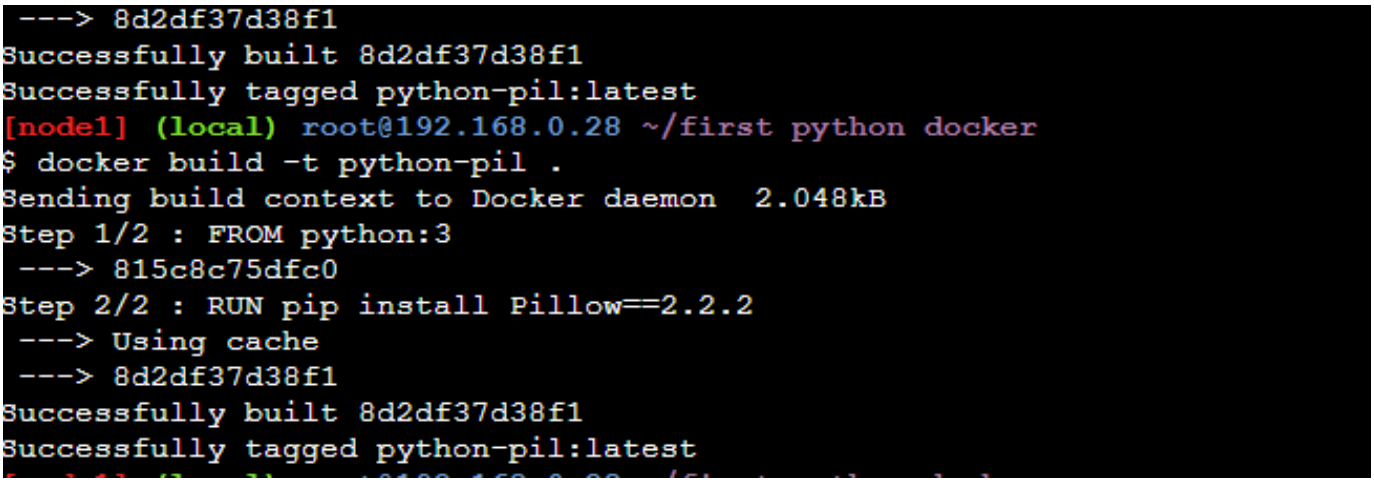
```
[node1] (local) root@192.168.0.28 ~/first python docker
$ vim Dockerfile
```

Use vim or any other editor to edit the Dockerfile.

```
1 FROM python:3
2 RUN pip install Pillow==2.2.2
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --
```

This is how your Dockerfile would look where your base image will be a python3 image and we will install the PIL library.

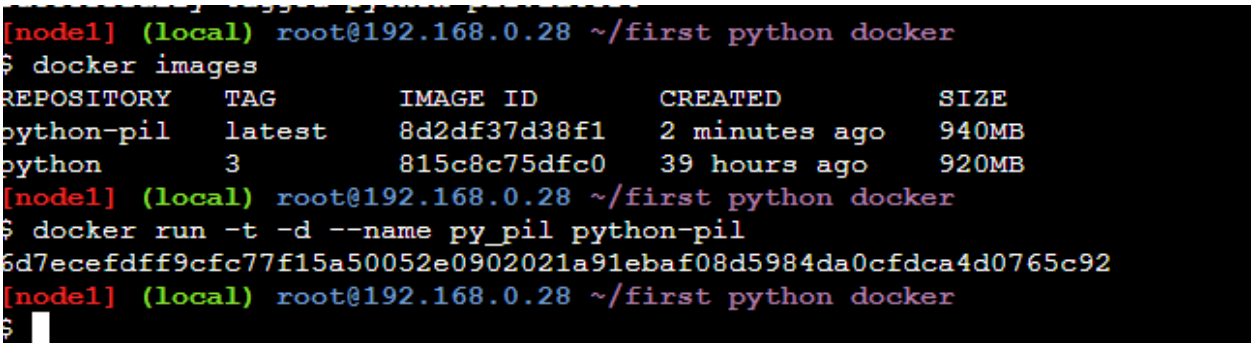
Build an image from the Dockerfile



The image is built. Now let us run the image to create a container.

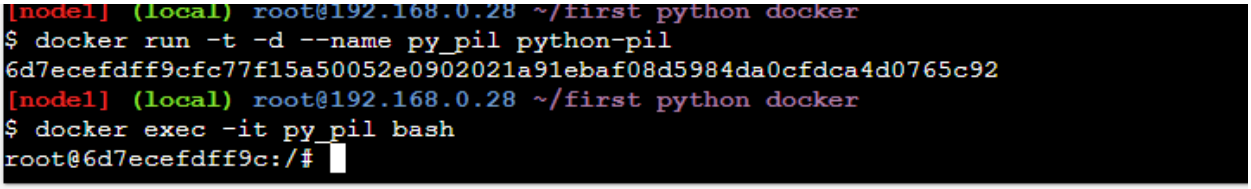
$docker build -t [docker image]

Run your image



Your container is UP. You can install anything here and run any code.

$docker run -t -d --name [tag] [image name]



You can enter inside the container with the command “docker exec -it py_pil bash”