1. *Installing Docker on Ubuntu*

The Docker installation package available in the official Ubuntu repository may not be the latest version. To ensure we get the latest version, we'll install Docker from the official Docker repository. To do that, we'll add a new package source, add the GPG key from Docker to ensure the downloads are valid, and then install the package.

First, update your existing list of packages:

```
sudo apt update
```

Next, install a few prerequisite packages which let apt use packages over HTTPS:

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

Then add the GPG key for the official Docker repository to your system:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Add the Docker repository to APT sources:

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
```

Next, update the package database with the Docker packages from the newly added repo:

```
sudo apt update
```

Make sure you are about to install from the Docker repo instead of the default Ubuntu repo:

```
apt-cache policy docker-ce
```

Finally, install Docker:

```
sudo apt install docker-ce
```

2. Docker Commands

Using docker consists of passing it a chain of options and commands followed by arguments. The syntax takes this form:

```
docker [option] [command] [arguments]
```

A. Working with Docker Images

Docker containers are built from Docker images. By default, Docker pulls these images from ***Docker Hub***, a Docker registry managed by Docker, the company behind the Docker project. Anyone can host their Docker images on Docker Hub, so most applications and Linux distributions you'll need will have images hosted there.

To check whether you can access and download images from Docker Hub, type:

```
docker run hello-world
```

The output will indicate that Docker in working correctly:

```
Output
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9bb5a5d4561a: Pull complete
Digest: sha256:3e1764d0f546ceac4565547df2ac4907fe46f007ea229fd7ef2718514bcec35d
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```

Docker was initially unable to find the hello-world image locally, so it downloaded the image from Docker Hub, which is the default repository. Once the image downloaded, Docker created a container from the image and the application within the container executed, displaying the message.

After an image has been downloaded, you can then run a container using the downloaded image with the run subcommand. As you saw with the hello-world example, if an image has not been downloaded when Docker is executed with the run subcommand, the Docker client will first download the image, then run a container using it.

To see the images that have been downloaded to your computer, type:

docker images

B. Running a Docker Container

The hello-world container you ran in the previous step is an example of a container that runs and exits after emitting a test message. Containers can be much more useful than that, and they can be interactive. After all, they are similar to virtual machines, only more resource-friendly.

As an example, let's run a container using the latest image of Ubuntu. The combination of the **-i** and **-t** switches gives you interactive shell access into the container:

docker run -it Ubuntu

C. Managing Docker Containers

After using Docker for a while, you'll have many active (running) and inactive containers on your computer. To view the **active ones**, use:

docker ps

To view all containers — active and inactive, run docker ps with the -a switch:

docker ps –a

To view the latest container you created, pass it the -l switch:

docker ps –l

To start a stopped container, use docker start, followed by the container ID or the container's name. Let's start the Ubuntu-based container with the ID ofd9b100f2f636:

docker start d9b100f2f636

To stop a running container, use docker stop, followed by the container ID or name. This time, we'll use the name that Docker assigned the container, which is sharp_volhard:

docker stop sharp_volhard

Once you've decided you no longer need a container anymore, remove it with the docker rm command, again using either the container ID or the name. Use the docker ps -a command to find the container ID or name for the container associated with the hello-world image and remove it.

docker rm festive_williams

D.Committing Changes in a Container to a Docker Image

When you start up a Docker image, you can create, modify, and delete files just like you can with a virtual machine. The changes that you make will only apply to that container. You can start and stop it, but once you destroy it with the docker ***rm*** command, the changes will be lost for good.

This section shows you how to save the state of a container as a new Docker image.

After installing Node.js inside the Ubuntu container, you now have a container running off an image, but the container is different from the image you used to create it. But you might want to reuse this Node.js container as the basis for new images later.

Then commit the changes to a new Docker image instance using the following command.

docker commit -m "What you did to the image" -a "Author Name" container_id repository/new_image_name

The **-m** switch is for the commit message that helps you and others know what changes you made, while **-a** is used to specify the author. The container_id is the one you noted earlier in the tutorial when

you started the interactive Docker session. Unless you created additional repositories on Docker Hub, the repository is usually your Docker Hub username.

For example, for the user **sammy**, with the container ID of d9b100f2f636, the command would be:

```
docker commit -m "added Node.js" -a "sammy" d9b100f2f636 sammy/ubuntu-nodejs
```

When you *commit* an image, the new image is saved locally on your computer. Later in this tutorial, you'll learn how to push an image to a Docker registry like Docker Hub so others can access it.

E. Pushing Docker Images to a Docker Repository

The next logical step after creating a new image from an existing image is to share it with a select few of your friends, the whole world on Docker Hub, or other Docker registry that you have access to. To push an image to Docker Hub or any other Docker registry, you must have an account there.

This section shows you how to push a Docker image to Docker Hub. To learn how to create your own private Docker registry, check out How To Set Up a Private Docker Registry on Ubuntu 14.04.

To push your image, first log into Docker Hub.

```
docker login -u docker-registry-username
```

You'll be prompted to authenticate using your Docker Hub password. If you specified the correct password, authentication should succeed.

Then you may push your own image using:

```
docker push docker-registry-username/docker-image-name
```

To push the ubuntu-nodejs image to the **sammy** repository, the command would be:

```
docker push sammy/ubuntu-nodejs
```

The process may take some time to complete as it uploads the images, but when completed, the output will look like this:

3. Conclusion

In this tutorial you installed Docker, worked with images and containers, and pushed a modified image to Docker Hub. Now that you know the basics, explore the other Docker tutorials in the DigitalOcean Community.

4. References

https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04