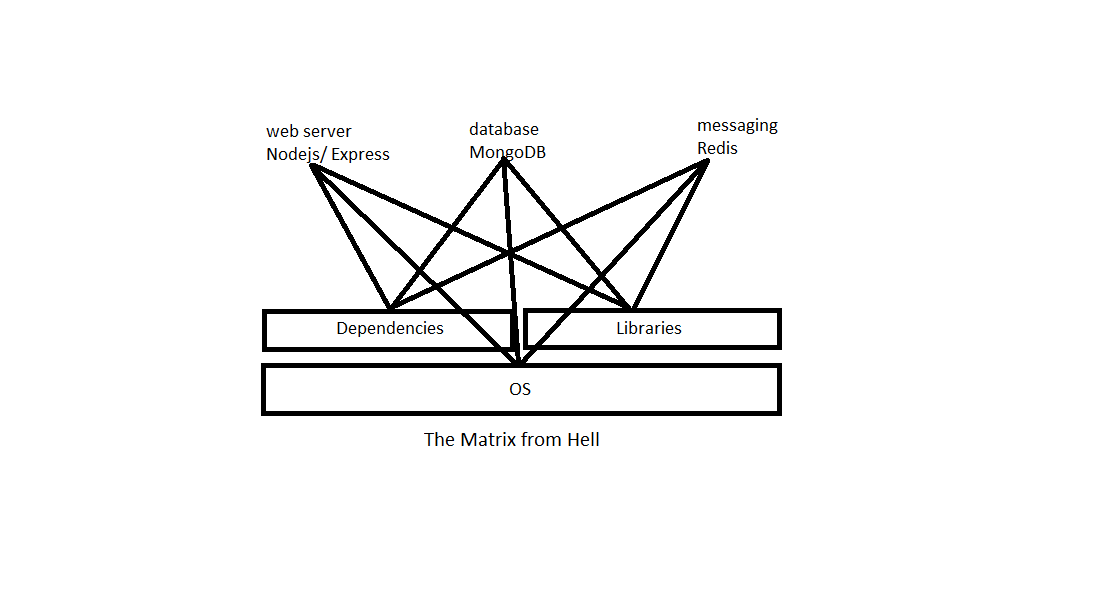
### Docker for the absolute beginners

What is Docker?

Docker is a software platform for building applications based on containers—small and lightweight execution environments that make shared use of the operating system kernel but otherwise run in isolation from one another.

Why do you need Docker?

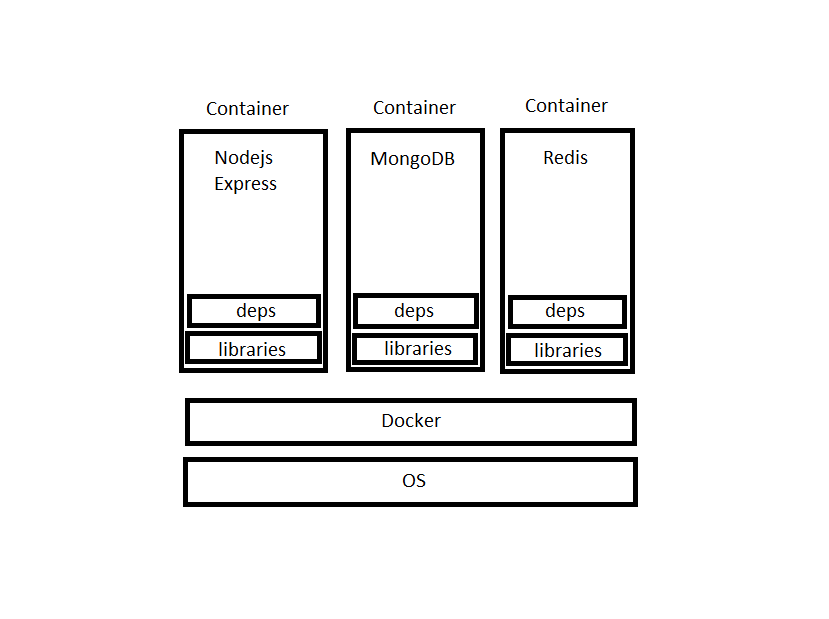
Let’s say you want to build an application using various technologies. For example, Nodejs and Express for the web server, MongoDB for the database and Redis for the messaging system. First of all, their compatibility with the underlying operating system (OS) is an issue. You have to make sure all the different services are compatible with the OS. Secondly, you have to make sure all the services are compatible with the libraries and dependencies of the OS. This compatibility matrix issue is referred to as “The Matrix from Hell”.



Read more at, <https://nathanleclaire.com/blog/2014/03/22/what-is-this-docker-thing-that-everyone-is-so-hyped-about>

Another thing to mention is that every time a new developer is onboard, they will have to follow a large set of instructions and run hundreds of commands to setup their environment. All of this will make the development, building and shipping process really hard.

What does Docker do?

With docker you will be able to run each component in its own container with its own dependencies and libraries. 

Once you’re done building the docker configuration file, anyone can get started with a **docker run** command irrespective of which operating they are on. They just have to install docker.

What are Docker images?

A Docker image is a read-only template that contains a set of instructions for creating a container. It provides a convenient way to package up applications and preconfigured server environments, which can be used for personal projects or shared publicly with other Docker users. Lots of programs have been dockerized already. You can find the images on docker hub.

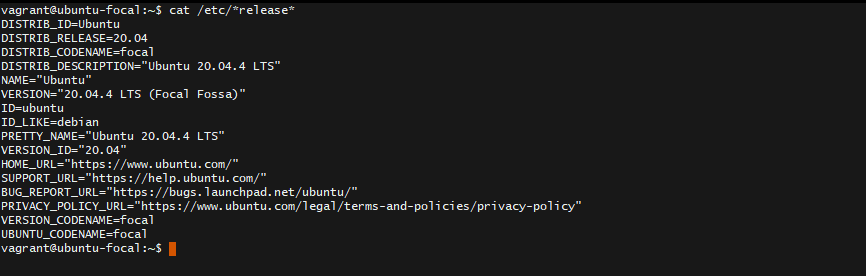
Images can be created with a Dockerfile which provides the specifications for creating a Docker image.

What are containers?

A container is a running instance of an image. It is a completely isolated environment. It has its own services, processes, network interfaces… just like virtual machines except they share the same OS kernel.

On this post, we will be learning docker from the basics by creating a simple web application. You will also be able to deploy your own image to docker hub. So, follow along.

I am on a Linux machine. The OS details are listed below.



Let’s start by installing docker.

Go to <https://docs.docker.com/engine/install/> and choose your OS. In my case, Ubuntu.

First, it will ask you to uninstall if you have an older version of docker already installed.

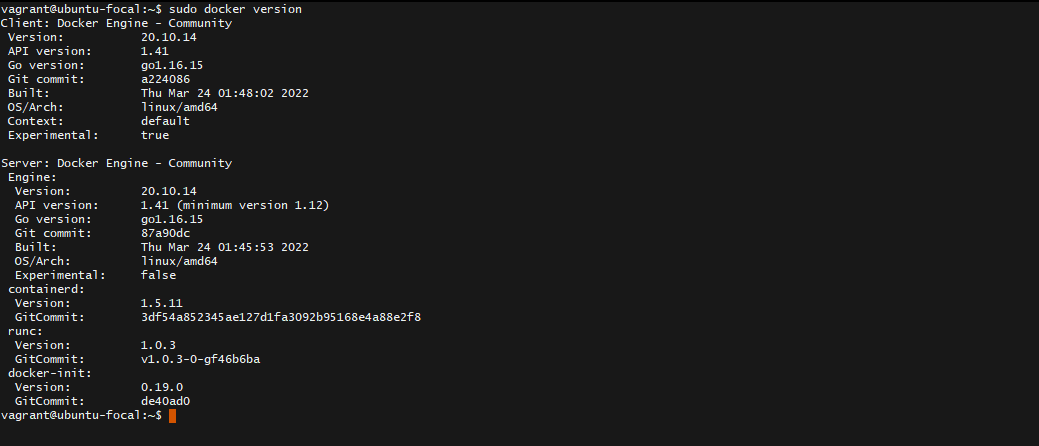
Now, there are two ways to install docker. For now, we will use the convenience script to install Docker. Scroll down to the **Install using the convenience script** section and follow along the installation.

curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh

The 1st command downloads the get-docker.sh shell script and the 2nd one executes it.

Now, if we type **sudo docker version** in terminal, the following output will be shown.



Congrats, you are done installing docker. The next step is to create a Dockerfile.

Open a terminal and navigate to your desktop folder. Then clone the following github repository <https://github.com/MyoniM/simple-web-app>. Now navigate to the cloned folder. It should contain an app.py file.

git clone <https://github.com/MyoniM/simple-web-app>

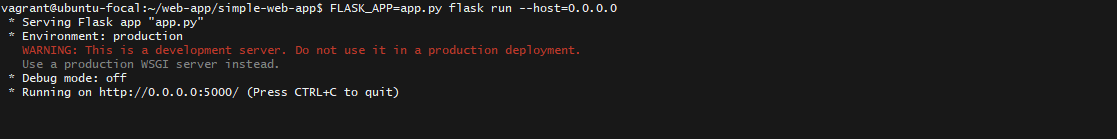
Before creating the Dockerfile, let us start the app.

The first step is downloading package information from all configured sources from our Linux repository list. To do so, type the command **sudo apt-get update** and wait for it to finish. This command might defer between operating systems.

Once that finishes, we’ll install **python3** and **pip** package manager. Run the command **sudo** **apt-get install -y python3 python3-pip**.

The next step is to install the python **flask** module, since the app uses flask as a server. Run **pip install flask** or **sudo apt install python3-flask**

Once that finishes, run the app with **FLASK\_APP=app.py flask run --host=0.0.0.0**. You should see an output like this.



Open another terminal and check if the API is working by typing the command **curl** [**http://0.0.0.0:5000**](http://0.0.0.0:5000). If it outputs a **Welcome!** text on the terminal,that means the server is working correctly.

Now, let’s create a Dockerfile. First, create the file with a name Dockerfile by typing the command **touch Dockerfile**.

Now type **history** to get the list of commands you previously typed to setup the app manually.



Open the Docker file with your preferred text editor. I will use **vi**. If you are also using vi, press **I** to go to insert mode.

The first line is **FROM ubuntu**. All Dockerfile must start from another image. In this case from the ubuntu image.

Then write what the image has to do when run. Write the commands as if you are setting up the server yourself. What was the first step when we set up the server? Run **sudo apt-get update.** We write these commands to the Dockerfile as shown below.

**RUN apt-get update**

**RUN apt-get install -y python3 python3-pip**

**RUN pip install flask**

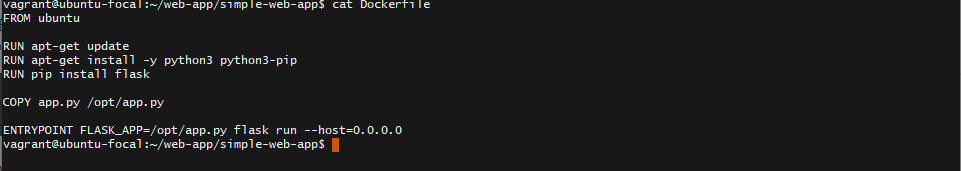
Earlier, we run the app by using the command **FLASK\_APP=app.py flask run --host=0.0.0.0**. This was possible because we were in the same directory as the app.py file. We are going to create an image from this Docker file. So, we need to copy the app.py file to a directory in the image when built. We do this by appending the following line,

**COPY app.py /opt/app.py**,to the Dockerfile. This command copies the app.py file to the **/opt** directory when the image is built.

Finally, we write what the image has to do when run. We can do this by appending the following command to the Dockerfile,

**ENTRYPOINT FLASK\_APP=/opt/app.py flask run --host=0.0.0.0**

We are all set up. If you’re using VI editor, press the **esc** character on your keyboard, write **:wq** and press enter to save and exit the file. However, if you’re using nano or similar text editor, press **ctrl + o** to save and **ctrl + x** to exit. The final result should be like below.

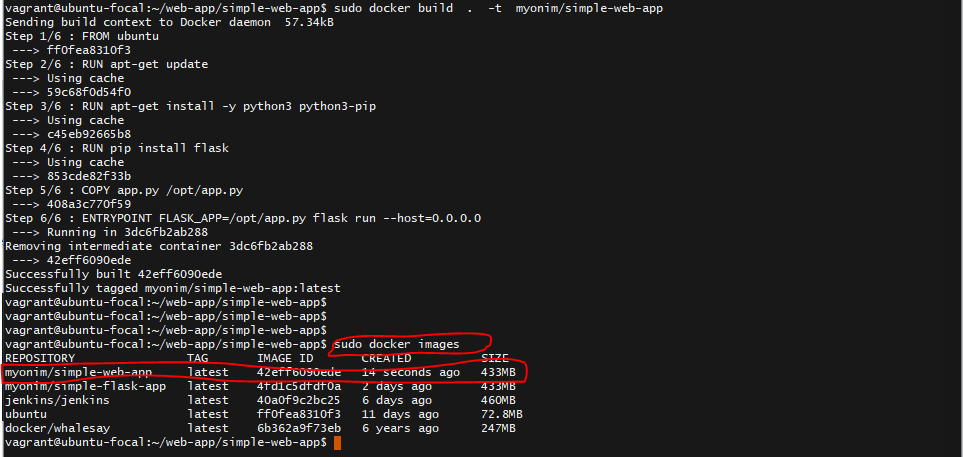


Time to build the image.

To build an image using a Dockerfile, run this command

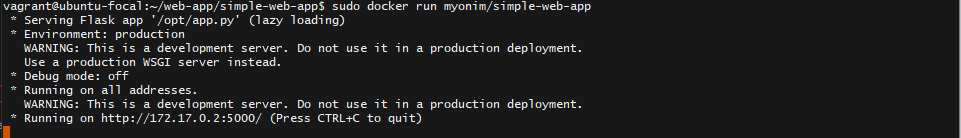
**docker build . -t myonim/simple-web-app**

**myonim** is my docker hub username, it is necessary if you want to deploy on docker hub repo. Go and sign up on <https://hub.docker.com>.



Type **sudo docker images** to check if the image is built and present. You are all set up, we can go and run our image by typing

**sudo docker run myonim/simple-web-app**



Perfect, we successfully run our image. It gave us a URL in which we can use to access the server at **172.17.0.2:5000**.

Open another terminal and check if the API is working by typing **curl** [**http://172.17.0.2:5000**](http://0.0.0.0:5000). If it says **Welcome!** that means it is working correctly.

You can see all running containers by typing

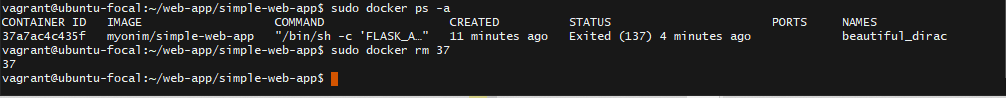
**sudo docker ps**

dockerps

I you want to stop the container, type **sudo docker stop id**, provide only first few characters from the id or the container name. If it returns the id or the container name, it is successful.

docker stop

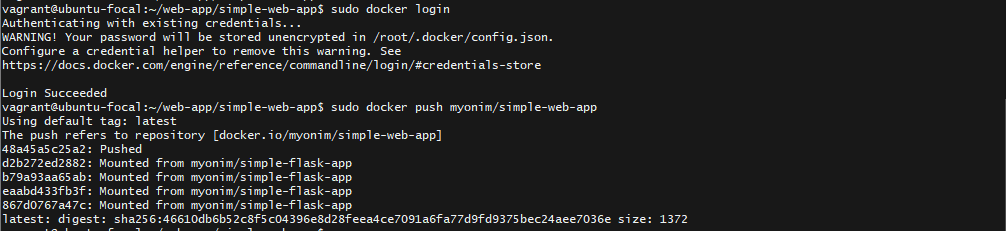
But, if you run **sudo docker ps -a,** you still can the container. It is seating idle consuming resources. so you should remove it by typing **sudo docker rm id/name**



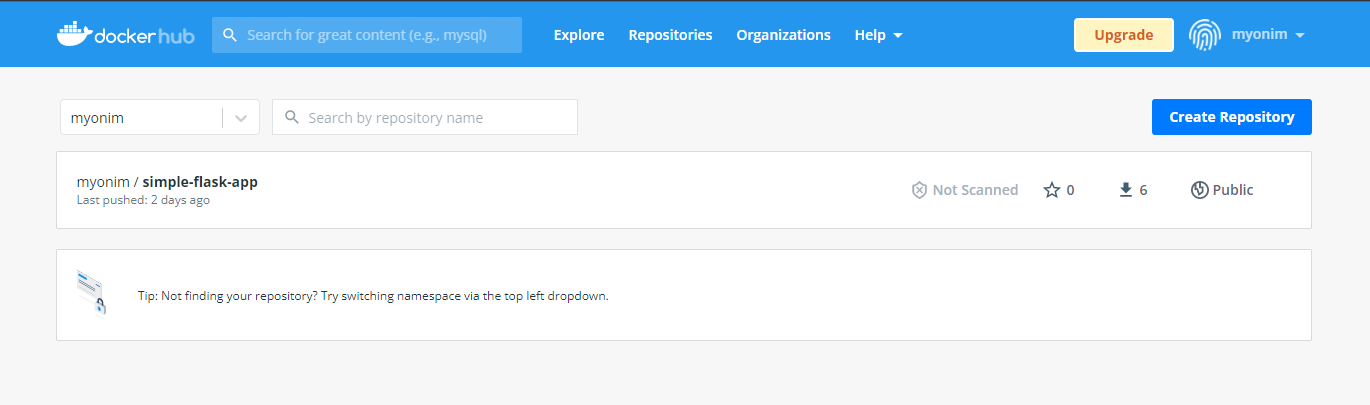
Finally, we can now deploy our image to docker hub.

After creating account on docker hub, go to your terminal and login. It will ask you for credentials.

After authenticating, use **sudo docker push imagename** to push your image.



If you go to your docker hub account you will see the following screen.



Networks in Docker

Docker Compose

Docker Concepts in Depth

Docker for Windows/Mac

Docker Swarm

Docker vs Kubernetes