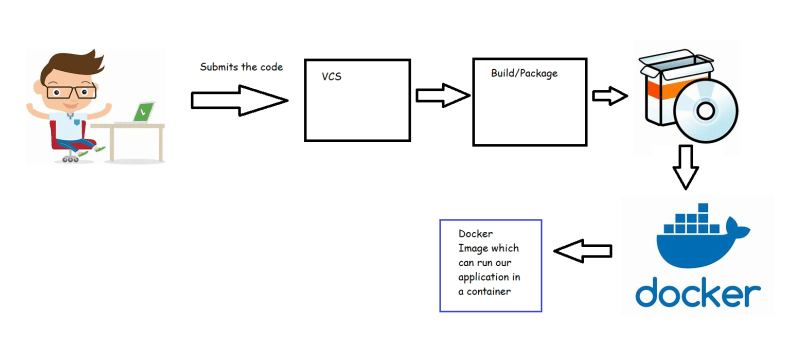
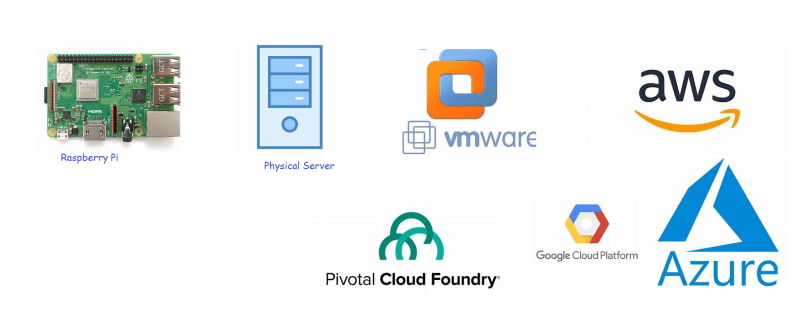
Docker Image

We need to package the applications developed in our organization as docker images 

Advantages of docker images:

* + Running an app on docker will give more priority to CPU and other resources consumed by application rather than os
  + Cost Reduction: On one host we can run multiple applications seamlessly which reduces hardware costs and os costs

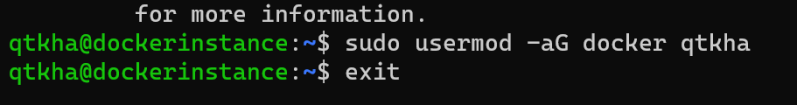
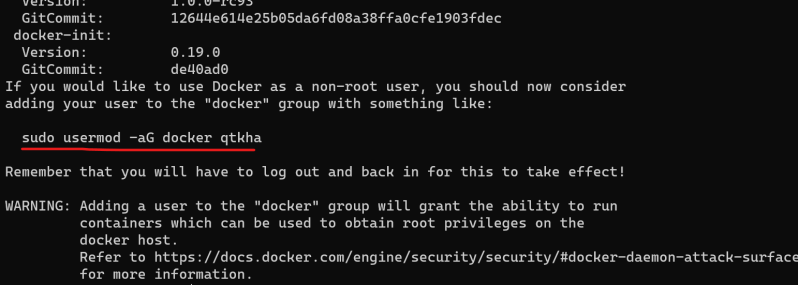
Docker is extremely portable; Docker container literally runs anywhere

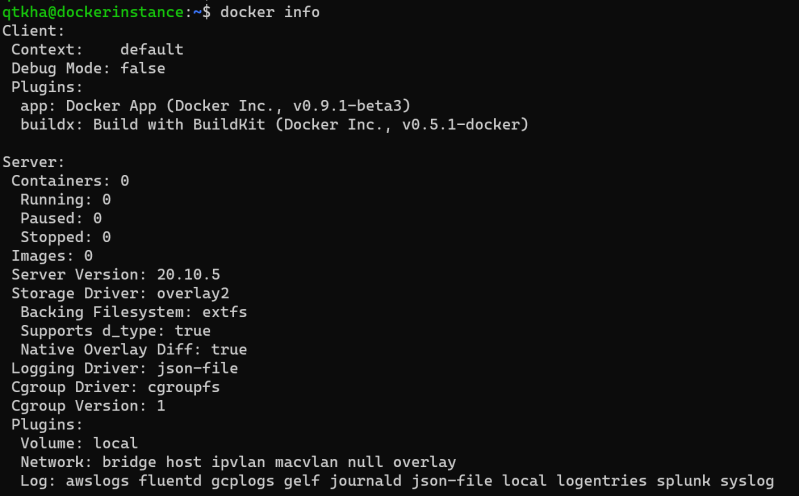
Installing Docker by Shell script

sudo apt update

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

sh get-docker.sh



Exit and relogging and execute the command docker info 

First look at building docker images

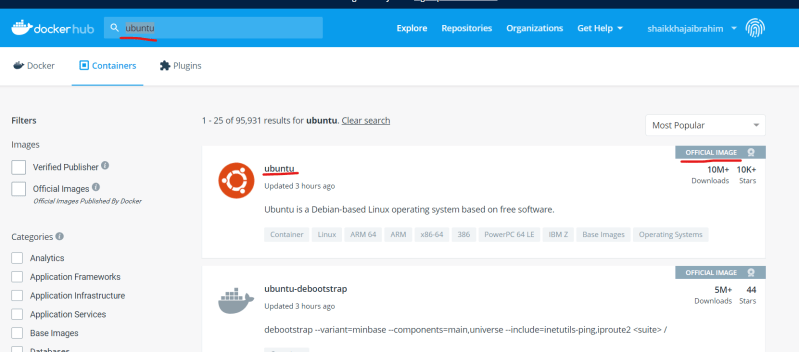
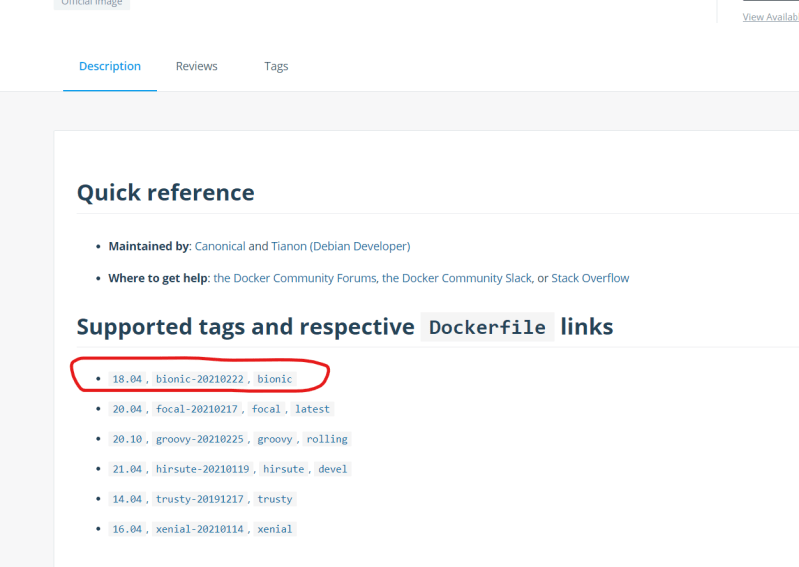
* Let’s take the lampstack as an example (URL: https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-ubuntu-18-04)
* First step is to make a note of all the steps required to install/configure your application manually on linux/windows machine
* Then try to create a Dockerfile which consists of instructions to install/configure your application
* Dockerfile is a text file which consists of set of Instructions to build a docker image
* Let’s try to make a note of steps to manually create a lamp server on ubuntu server

sudo apt update

sudo apt install apache2 -y

sudo apt install php libapache2-mod-php php-mysql php-cli -y

echo '<?php phpinfo(); ?>' | sudo tee /var/www/html/info.php

* Basic docker image creation process is
  + choose a base image
  + add your installation/configuration steps
  + create the new image
* To choose a base image navigate to (Link: https://hub.docker.com/search?q=&type=image)
* In docker the version of image is referred as tag (Link: https://hub.docker.com/\_/ubuntu)
* Since our steps are based on ubuntu 18.04 lets choose the tag 
* Let’s try to manually check if the image will work or not
* So lets try to create a container with the ubuntu image and 18.04 tag

docker container run -it <image> /bin/bash

* In docker convention is <image-name>:<tag> => ubuntu:18.04

docker container run -it ubuntu:18.04 /bin/bash

* This command creates a ubuntu 18 container and you will be logged in to the container as a root user
* For commands start using cheatsheet (https://www.docker.com/sites/default/files/d8/2019-09/docker-cheat-sheet.pdf)
* In the container we have executed the following steps

apt update

apt install apache2 -y

apt install php libapache2-mod-php php-mysql php-cli -y

echo '<?php phpinfo(); ?>' | tee /var/www/html/info.php

service apache2 start

The way to write Dockerfile contents

FROM ubuntu:18.04

RUN apt update

RUN apt install apache2 -y

RUN apt install php libapache2-mod-php php-mysql php-cli -y

RUN echo '<?php phpinfo(); ?>' | tee /var/www/html/info.php

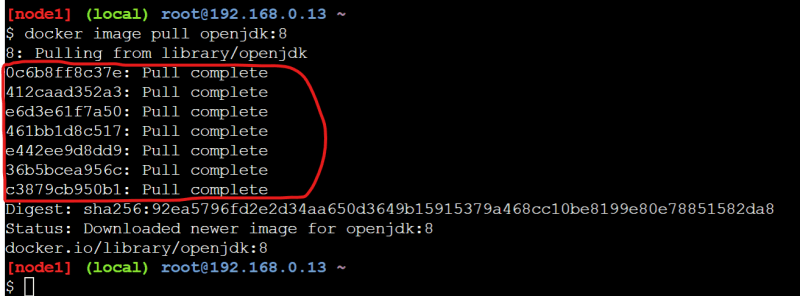
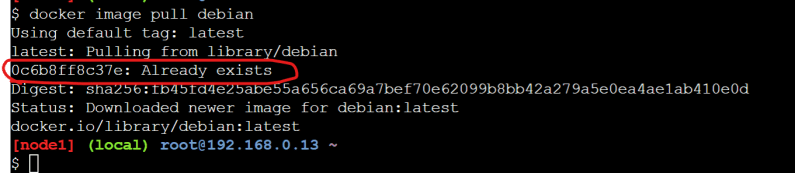
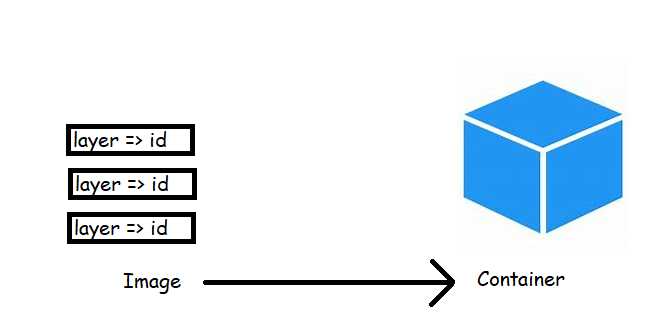
RUN service apache2 start

EXPOSE 80

Let’s look at ubuntu Dockerfile created by community

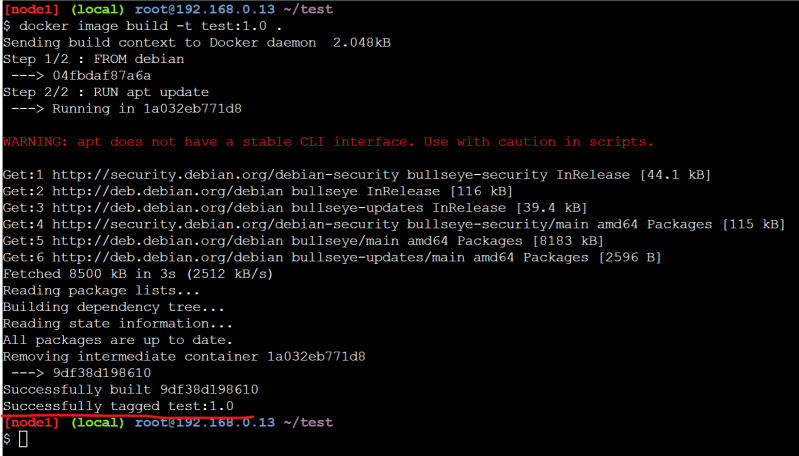
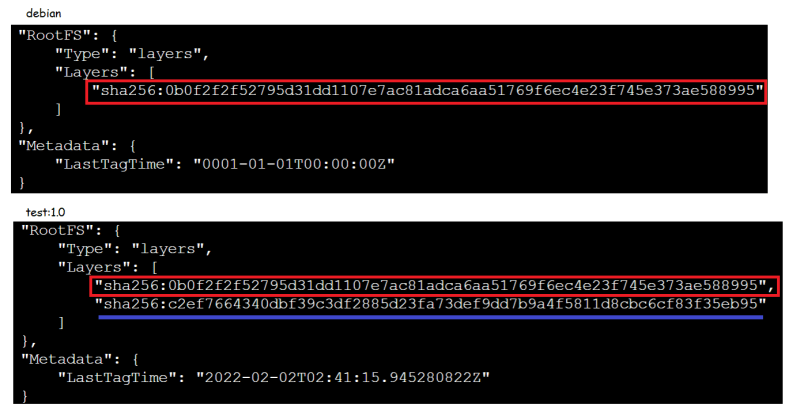


## Docker Images

* Docker images are required to create containers.
* From one Docker image, we can create multiple containers
* Images are build-time constructs and containers are run-time constructs.
* Let pull some images and observe the stuff as shown below  
    
  
* Docker image is collection of Image Layers  
  
* When we build the image we use some base image and on top of this we do add some stuff related to our applications. Let me write a simple Dockerfile and create a image from this

FROM debian

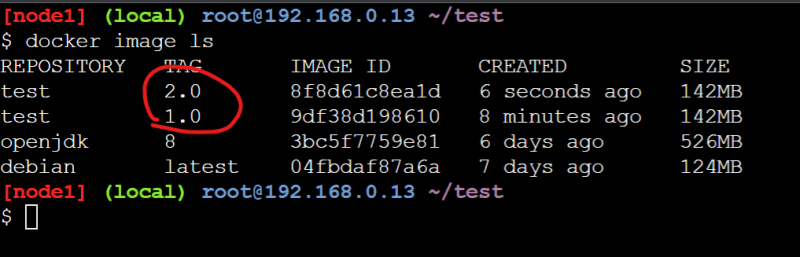
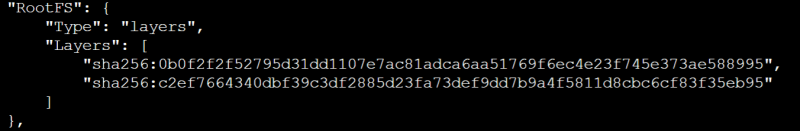
RUN apt update

  
\* Now lets inspect the debian image docker image inspect debian and test image docker image inspect test:1.0  
  
\* The test:1.0 uses the layer from debian and adds a new layer this new layer is created as the result of the RUN statement (RUN apt update).  
\* Now lets build a new docker image test:2.0 with the following Dockerfile and inspect the image

FROM debian

RUN apt update

CMD ["echo", "helloworld"]

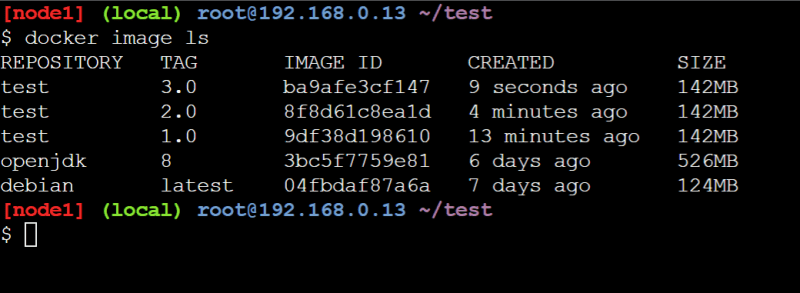
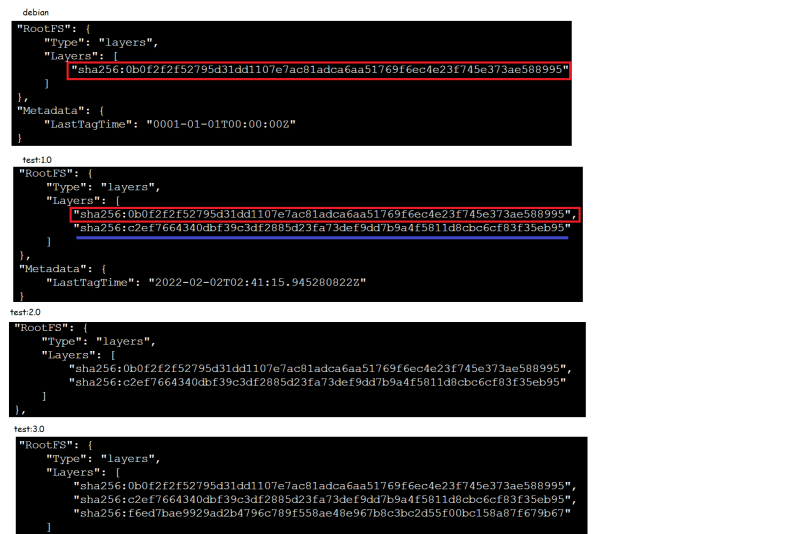
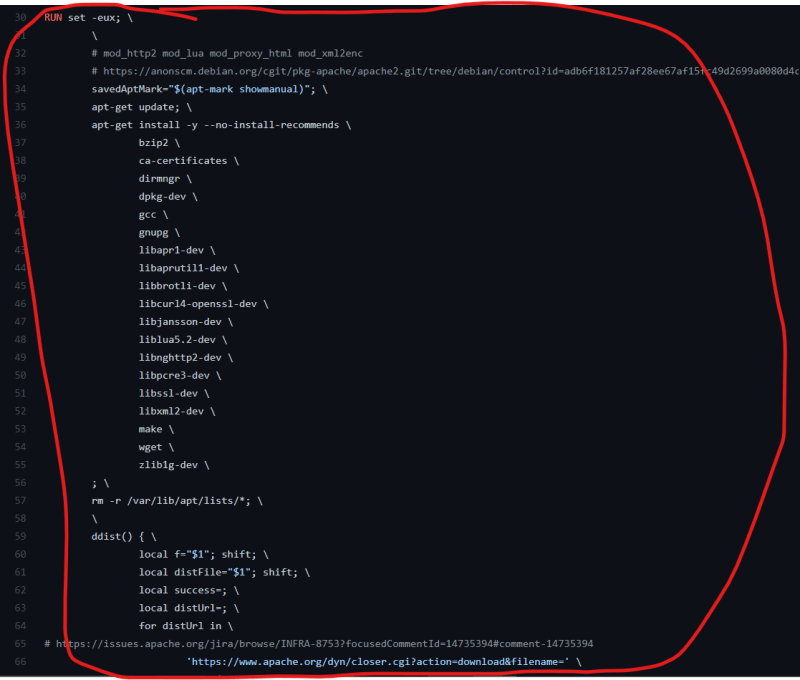
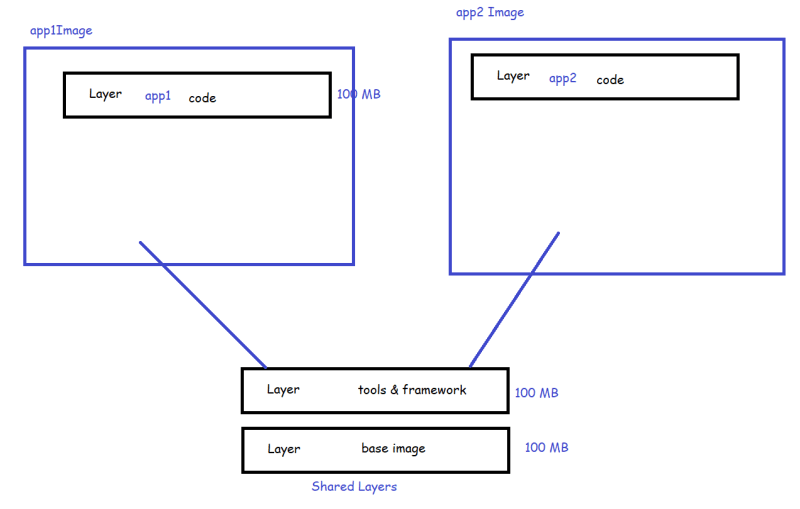
  
\* Lets inspect test:2.0  
  
\* Lets update the Docker file and build the docker image test:3.0

FROM debian

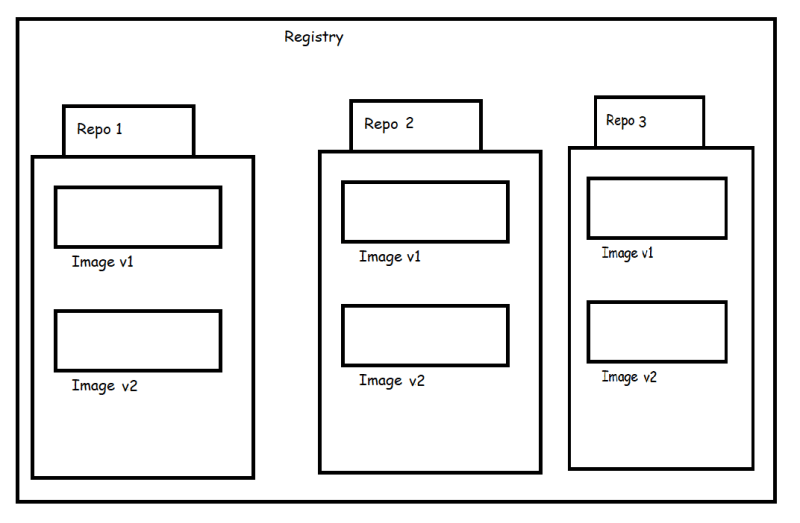
RUN apt update

ADD test.txt /

CMD ["echo", "helloworld"]

  
\* As we have observed so far, whenever we make changes that leads to some changes in the file system (RUN, ADD) a new layer is getting created  
  
\* Creating too many layers is not considered as good practice so you would often see Docker files with long RUN statements RUN statement1 && statement2  
  
\* The Layers get shared across images  
  
\* We would write Dockerfiles for our applications and create the docker images.

## Docker image Registry

* We can store Docker images centrally in image Registries  
  
* The Default Registries is DockerHub.
* There are many other Registries to Store docker images
* Cloud Registries:
  + Azure Container Registry
  + AWS ECR
  + Docker Hub (Public and Private)
  + And many other
* Hosted Registries
  + Artifactory/Jfrog
  + We can also use Docker Registry Image to store our images

CE vs EE

Docker CE (Community Edition) is the simple classical OSS (Open Source Software) Docker Engine.

Docker EE (Enterprise Edition) is Docker CE with certification on some systems and support by Docker Inc.

