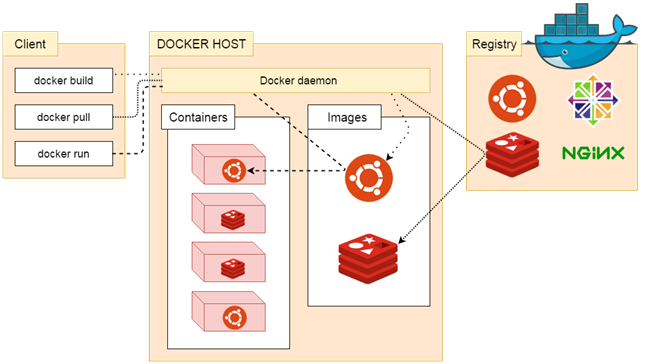
Docker follows client-server architecture. Its architecture consists mainly three parts.

1) **Client:** Docker provides Command Line Interface (CLI) tools to client to interact with Docker daemon. Client can build, run and stop application. Client can also interact to Docker\_Host remotely.

2) **Docker\_Host:** It contains Containers, Images, and Docker daemon. It provides complete environment to execute and run your application.

3) **Registry:** It is global repository of images. You can access and use these images to run your application in Docker environment

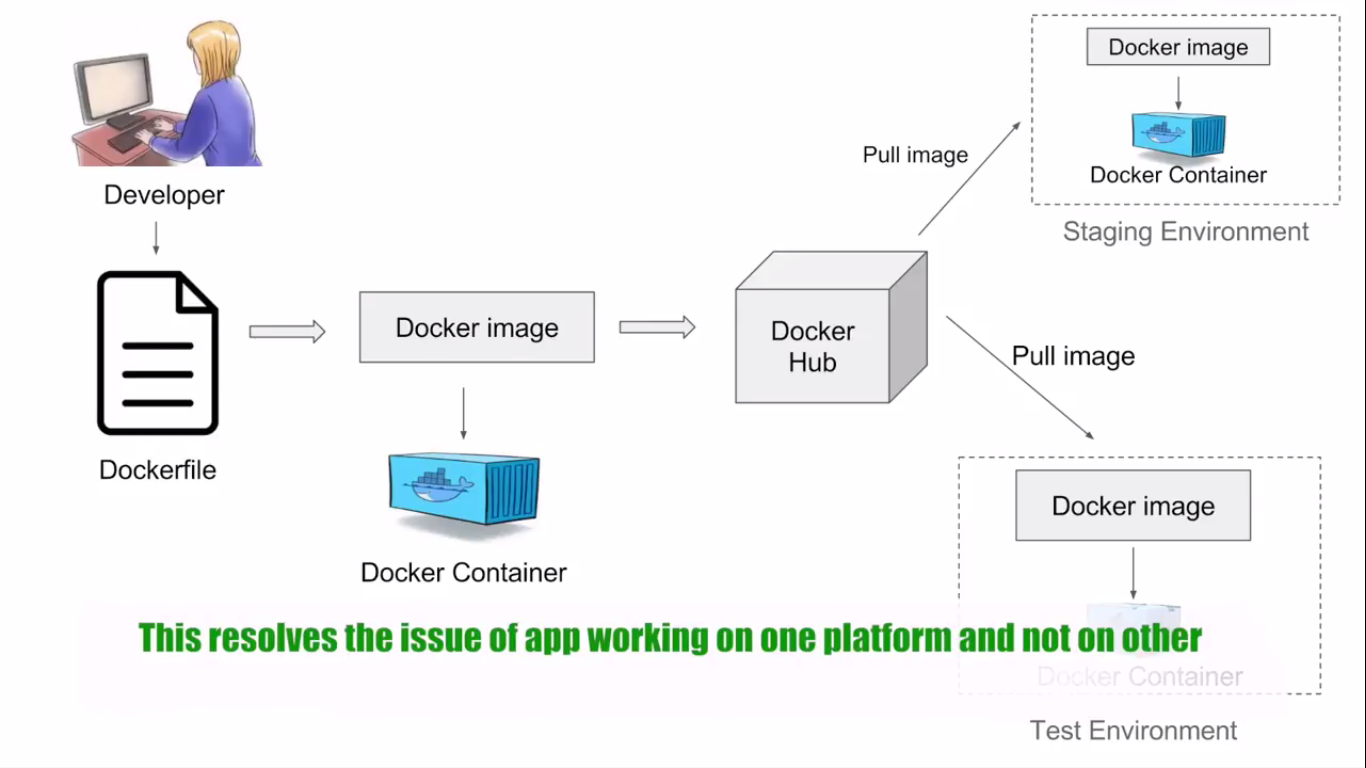


**Docker Work Flow**

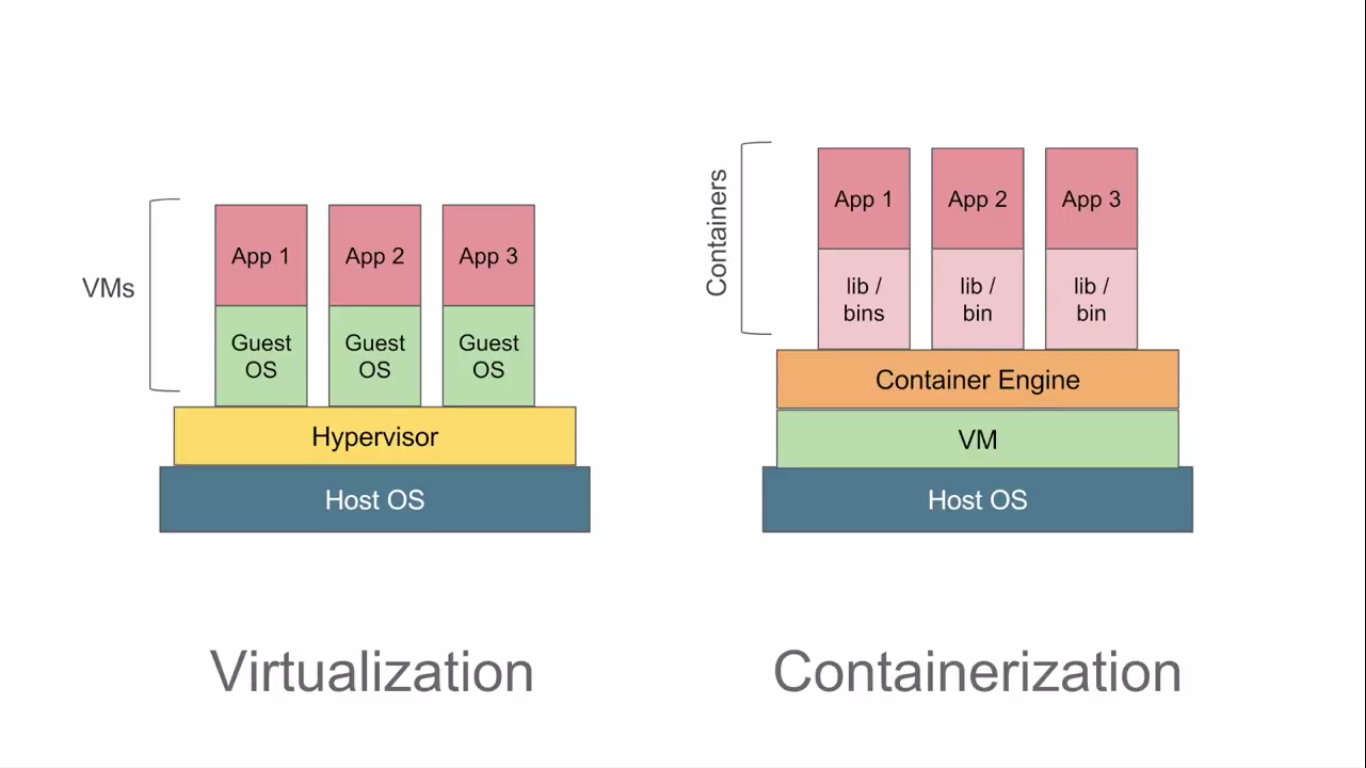
1) Developer will define all the application and its dependency and requirements in file, that file will call "docker file".

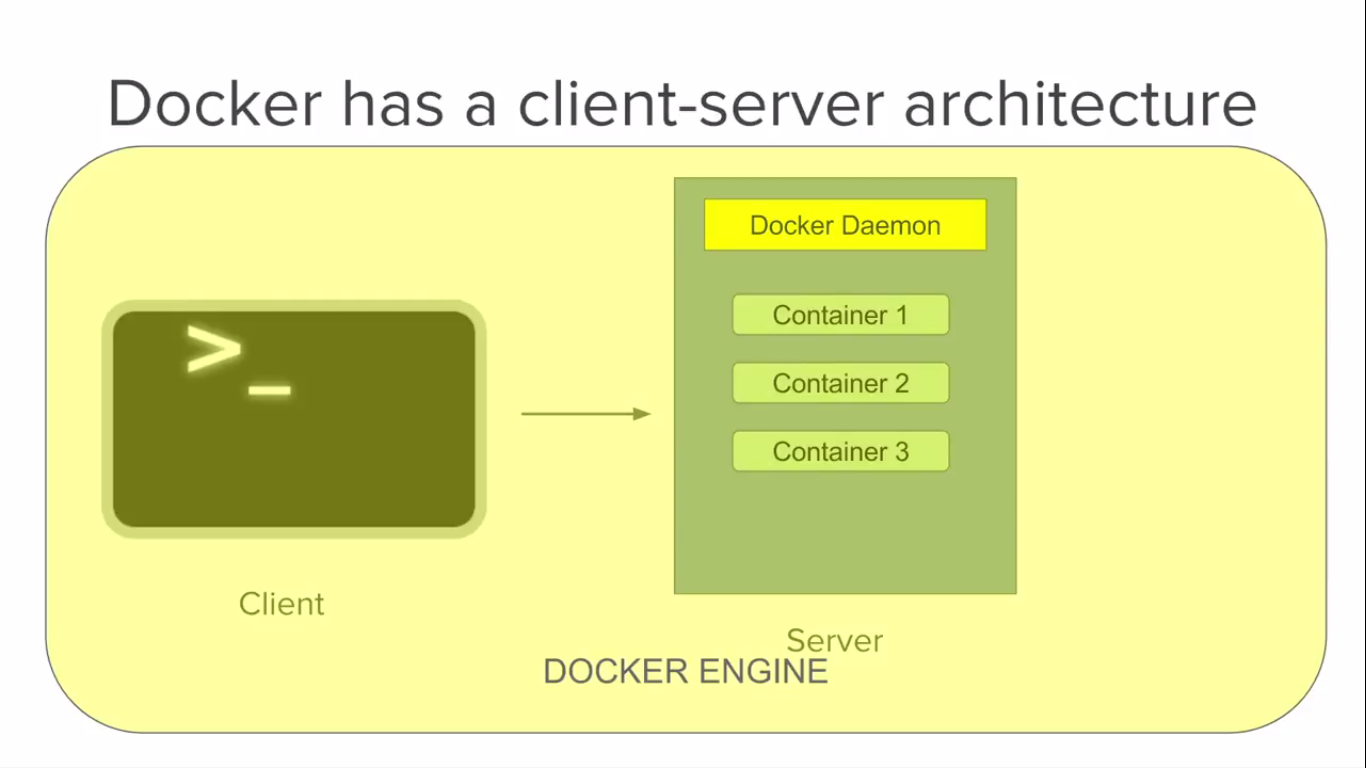
2) Docker file will use create a docker image (). in the Docker image will have all the application it's requirements and dependencies.

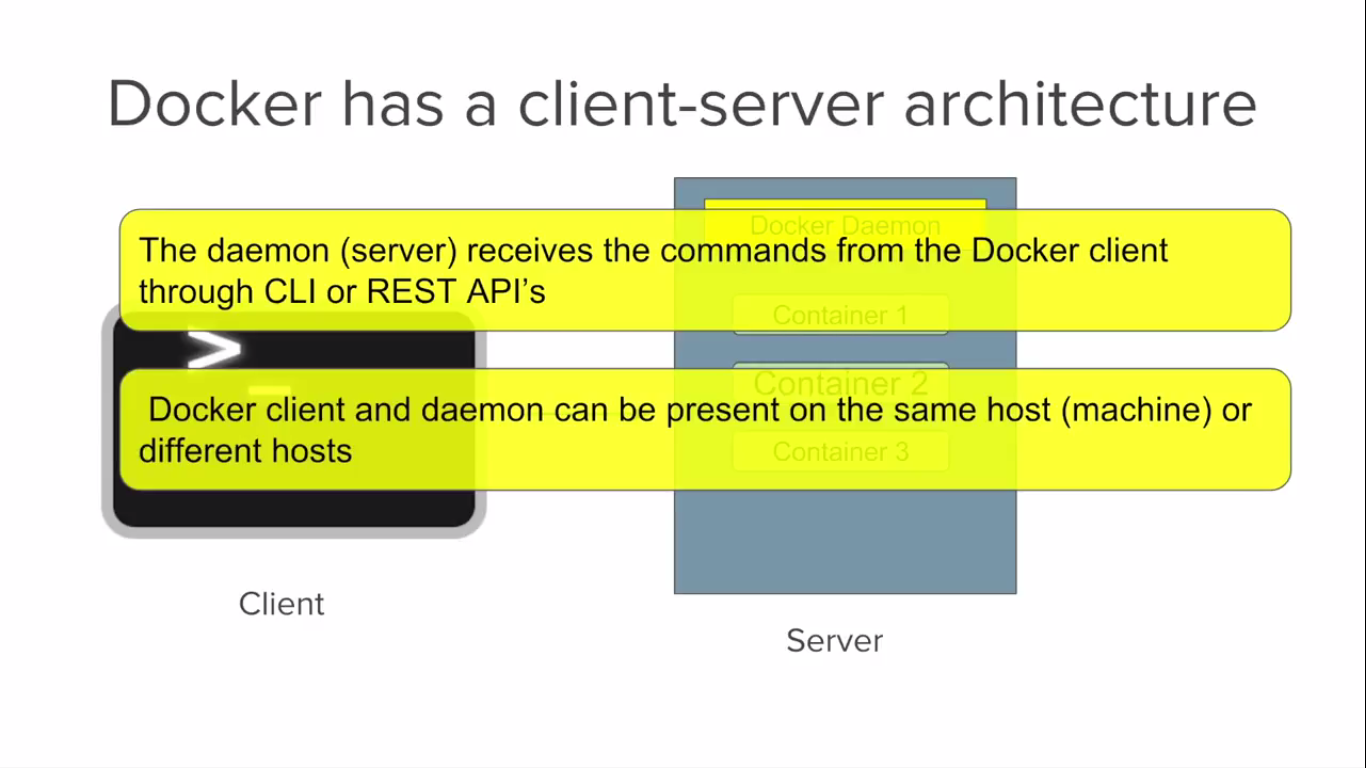
3) when we run docker image, then we will get docker container.



**Virtualization VS Containerization**

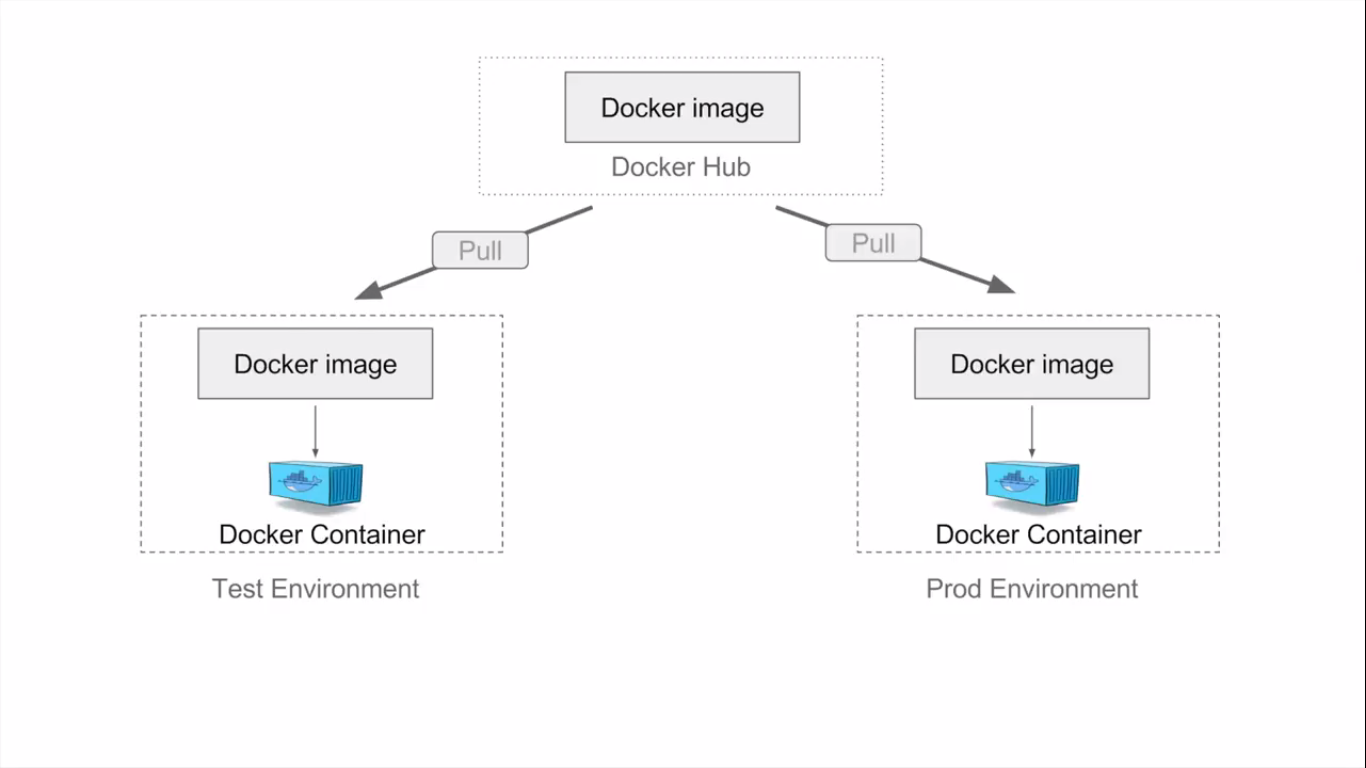


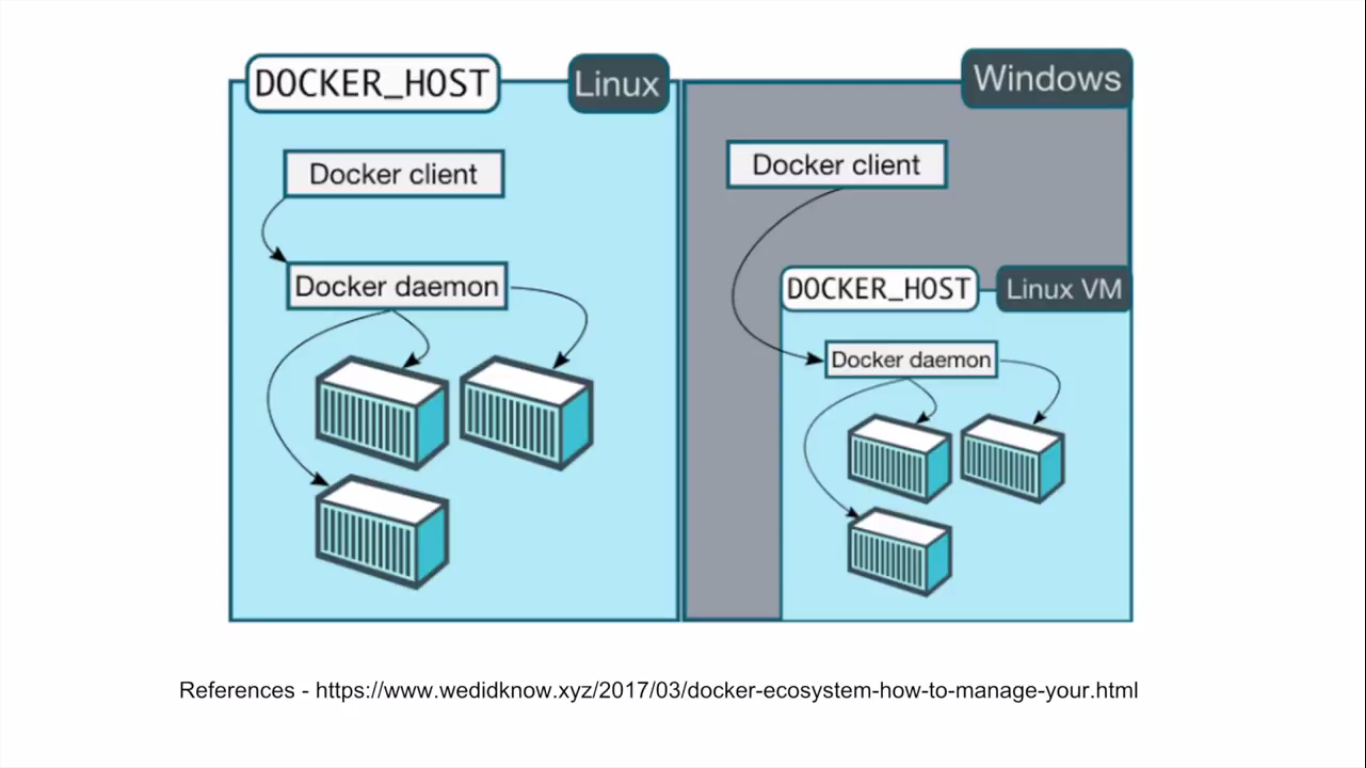




**Benefits**

* Build application once. An application inside a container can run on any system that has docker Docker installed. There is no need to build and configure app multiple times on different platforms.
* Create Docker image and put into “**docker hub**” or put in any repository and we can pull this image on any environment like test. Run the image for create a docker container and our application will run docker container. We can do in the same way for other environments like production. If your application is run in test environment, that will run on production also without any problem.
* The same environment identical in the same docker container.





* Portability - Docker container can run any platform. It can run local system, AWS, google cloud, virtual box etc…
* Isolation – with docker with every application works in its own container, it does not interfere other application running on the same system
* So, multiple container can run in the same system without interference

**Details**

* Docker Hub is a registry service on the cloud that allows you to download Docker images that are built by other communities
* In Docker, everything is based on Images. An image is a combination of a file system and parameters
* Containers are instances of Docker images that can be run using the Docker run command. The basic purpose of Docker is to run containers.
* # docker top ContainerID - With this command, you can see the top processes within a container.
* Docker also gives you the capability to create your own Docker images, and it can be done with the help of Docker Files. A Docker File is a simple text file with instructions on how to build your images

**Install Docker in Linux**

**Prerequisites:** 1) Connect to Linux, 2) Install Docker, 3) Run Docker

* Check operating system, kernel version using command “# uname -r”
* Update packages in cent OS “# yum -y update”
* Run **“# yum install -y docker**”
* Check docker # docker --version

Docker version 1.12.6, build 88a4867/1.12.6

* Run the Docker service **# service docker start**

Redirecting to **/bin/systemctl start docker.service**

* Check the service using **# docker info**
* Get Docker images **# docker images**
* Get Running docker **containers # docker ps (OR) # docker ps -a**
* **# docker run hello-world**

(**Res**

Unable to find image 'hello-world:latest' locally

Trying to pull repository docker.io/library/hello-world ...

latest: Pulling from docker.io/library/hello-world

b04784fba78d: Pull complete

Digest: sha256:f3b3b28a45160805bb16542c9531888519430e9e6d6ffc09d72261b0d26ff74f

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

3. The Docker daemon created a new container from that image which runs the

executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it

to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://cloud.docker.com/

For more examples and ideas, visit:

* <https://docs.docker.com/engine/userguide/> )
* “hello-world” is the image name. if it is not available in local system, it will look into online docker repository and pull from there and run and start the container.
* Search for docker images from docker repository. Example search for mysql docker image

**# docker search mysql**

* Next pull & run image it using syn:
* # docker run --name nameOfTheImage -e MYSQL\_ROOT\_PASSORD=ROOT -d mysql/mysql-server:5.7
* # docker run --name srinimysql -e MYSQL\_ROOT\_PASSORD=ROOT -d mysql/mysql-server:5.7
* Run mysql shall
* Docker exec -it srinimysql mysql -uroot -p
* Than it will ask password and will get mysql shall.
* Now go to mysql container : # docker exec -it srinimysql bash
* Now check mysql log in bash console : # cat /var/log/mysqld.log
* Stop the docker container : # docker stop image\_name/image\_id
* Remove docker image : # docker rmi -f image\_name/image\_id
* Remove docker conainer : # docker rm image\_name/image\_id

--------------------------------------------------------------------------------------------

**Installation method 2**

1. Install OS packages & updates using $**yum install -y yum-utils**
2. Use the following command to set up the stable repository. You always need the stable repository, even if you want to install builds from the edge or test repositories as well.

**$ yum-config-manager \**

**--add-repo \**

**https://download.docker.com/linux/centos/docker-ce.repo**

1. **Optional:** Enable the edge and test repositories. These repositories are included in the docker.repo file above but are disabled by default. You can enable them alongside the stable repository.

$ sudo yum-config-manager --enable docker-ce-edge

$ sudo yum-config-manager --enable docker-ce-test

1. Docker is available in two editions: **Community Edition (CE)** and **Enterprise Edition (EE)**.
2. Docker Enterprise Edition (EE) is designed for enterprise development and IT teams who build, ship, and run business critical applications in production at scale
3. Install the latest version of Docker CE, or go to the next step to install a specific version.

**$ sudo yum install docker-ce**

Docker is installed but not started. The docker group is created, but no users are added to the group.

On production systems, you should install a specific version of Docker CE instead of always using the latest. List the available versions. This example uses the sort -r command to sort the results by version number, highest to lowest, and is truncated.

$ yum list docker-ce --showduplicates | sort -r

docker-ce.x86\_64 17.09.ce-1.el7.centos docker-ce-stable

$ sudo yum install <FULLY-QUALIFIED-PACKAGE-NAME> (fully qualified package name is docker-ce-17.06.1.ce)

1. Start Docker.

**$ sudo systemctl start docker**

1. Verify that docker is installed correctly by running the hello-world image.

$ sudo docker run hello-world

1. Upgrade Docker CE

$yum -y upgrade

1. Create container from current terminal (now I am in cent OS root **[root@localhost ~]#**), from that terminal we are create container & attached to the current terminal (-t) using bellow command. After attached to the terminal we will interact (-i) with the created container.

$docker run -i -t <IMAGE> (OR) $docker run -it <IMAGE>

Ex: **docker run -it ubuntu**

- First it will check in local, if there is not available than it will pick from docker public repository and install the container. Now you automatically go to ubuntu OS. (**root@b4a9955d6ea8:/#)** check your host name and check OS using command

$ cat /etc/\*os\*rel\*

Now we are working on ubuntu. Host machine is centOS7, in that container is ubuntu. use “exit” come out from container means ubuntu.

1. $docker run -it <IMAGE> <entry point>
2. If we use this “docker run -it ubuntu” it will create different host name every time. If you want same than use “--name <name of your image>” parameter.

$docker run --name myubuntuhost -it <IMAGE> <entry point>

Ex: **$docker run --name myubuntuhost -it ubuntu /bin/bash**

If you want to go to ubuntu terminal use

$**docker start myubuntuhost**

$**docker attach myubuntuhost**

1. If run the container in background, if came out from container it will run in background.

**$docker run --name myubuntuhost -d ubuntu** (-d demonised mode )

Ex: $docker run --name myubuntuhost -d ubuntu bin/sh -c "while true; do echo hello world ; sleep 1; done"

(bin/sh -c "while true; do echo hello world ; sleep 1; done" command print hello world message every sec on terminal.

Now you are in host machine. If we want to see the container logs

**Start the container:**

$docker start myubuntuhost

Stop Container

$ **docker stop <container ID>**  
Now to restart the Docker container you can use:  
**$ docker restart <container ID>**

1. Check the list of process in container

**$docker top myubuntuhost**

1. Run the commands on container without enter into the container (run this commands from host machine)

**$docker exec myubuntuhost touch /tmp/srini.txt**

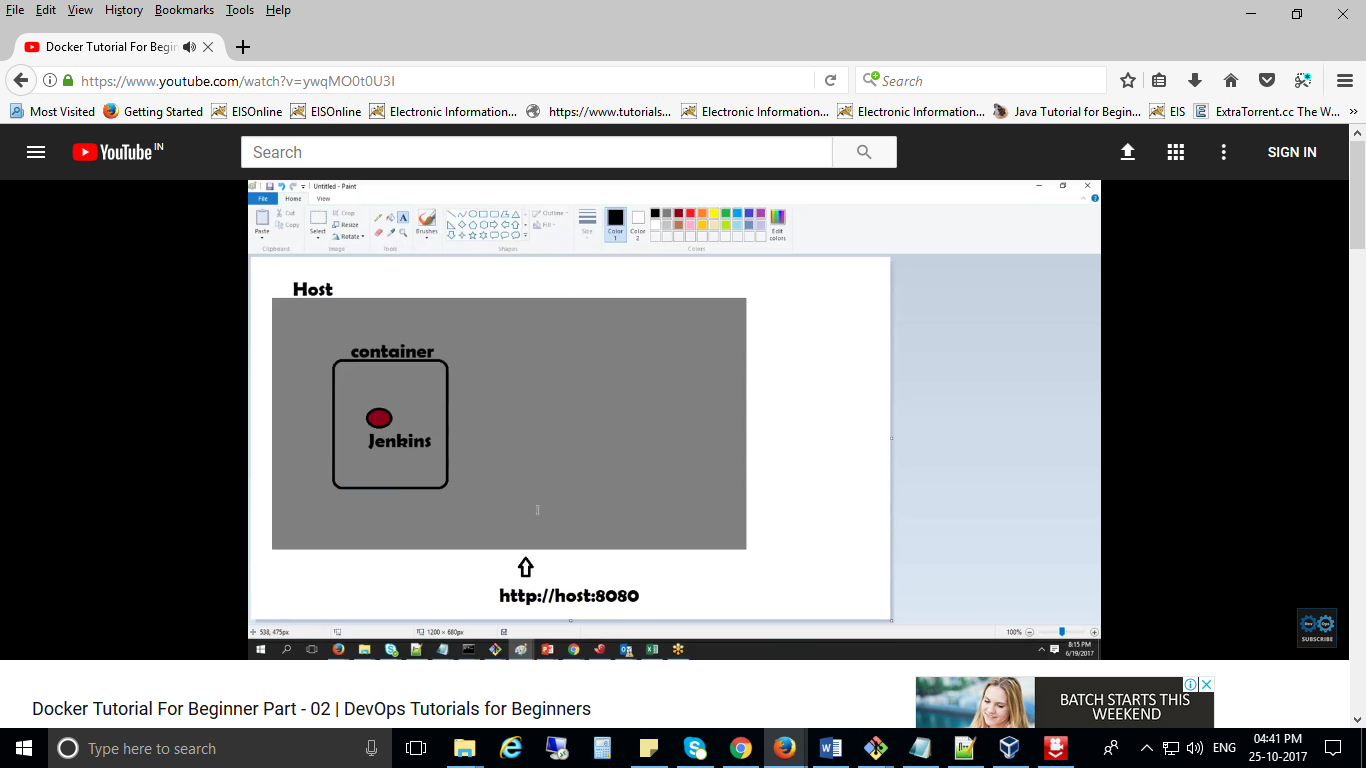
1. Get statistics information on container (here ubuntu)

**$docker stats myubuntuhost**

1. We can see each and every step (background also)

**$docker events**

**$docker events --filter event-attach**

1. 
2. Data Volumes : this is specially designed directory within one or more containers that bypasses the union file system. Volume can be shared and reused between containers. Declared volume using command

**$ docker run --name volcount -it -v /user/data ubuntu /bin/bash** (now in /user/ data folder created).

Now create one more container **testdatavol** and access **volcount** container “data” folder.

**$ docker run --name testdatavol -it –volumes-from volcount --privileged=true**

1. Share host folder to containers.

**$docker run -it -v /home/user/sharefolder:/user/data ubuntu /bin/bash**

1. Rename the docker name (reference name only not container id):

**$docker rename <src> <dest>**

1. Show available images: **$docker images**
2. Search for images **$docker search ubuntu**
3. Download image **$docker pull debian:jessie**
4. In ubuntu **$apt-get install vi**
5. Create or build images from our customised container (we installed our applications in this container). $docker commit <ourcontainername>

**$docker commit mytemplate**

Now it will create new image with image id (no name) using **mytemplate** container. Now we have to give name for image using “tag”

**$docker tag <new image id> <image name>**

Ex: $**docker tag b4a9955d6ea8 myimage**

Now run the image

**$docker run --it myimage /bin/bash**

1. Create Docker file. Using that file we can create customised container. Docker file name is “Dockerfile”. Write the code in Dockerfile.

------

FROM ubuntu

RUN apt-get update

RUN apt-get -y install vim

USER srini

WORKDIR /tmp/srini

RUN touch test.txt

ENV myname admin

--------

In above file first line must be actual image name using “FROM” here “ubuntu” is image name. after that update the ubuntu & install “vim” command, “-y” by default yes. After create docker file run that using build.

**$ docker build -t myubuntu2 .** (here . means current directory OR give path).

**$ docker run -it myubuntu2**

**$ docker history-it myubuntu2**

1. Docker file instructions:

**RUN**: Run the command when the container is being build.

**CMD**: specifies the command to run when the container is launched, if values are specified during the launch it will override the Dockerfile vale

Ex: CMD [“echo”,”Hello”]

$docker build -rm -t “admin/dockerfile”

$docker run -it “admin/dockerfile”

**ENTRYPOINT**: same as RUN, arguments we specified on the docker run command line will be passed as arguments to the command specified in the ENRTYPOINT

Ex: ENTRYPOINT [“echo”,”Hello”]

**WORKDIR**: by default it will take the root(/) directory, if we want do things OR steps in particular directory than use this.

Ex: WORKDIR /temp

RUN touch test.txt

Now create “test.txt” in “temp” folder, if we not use “WORKDIR /temp” than it will create “test.txt” file in root (/) directory.

We can also pass the directory from command line using -w

$docker run -it -w /user myubantu2 /bin/bash

**ENV**: set environment variables during the image build process

**USER**: specifies a user that the image should be run

Ex: USER <user name> , USER srini (now it will create srini user in container)

From command line: **$docker run -it -u srini myubuntu2 /bin/bash**

**VOLUME**: we can create sharing folder from docker file.

Ex: VOLUME [“/data”]

**COPY**: copy file from host machine to crating container

Ex: COPY test.txt /data/

**ADD**: it is used for copy zip, tar files

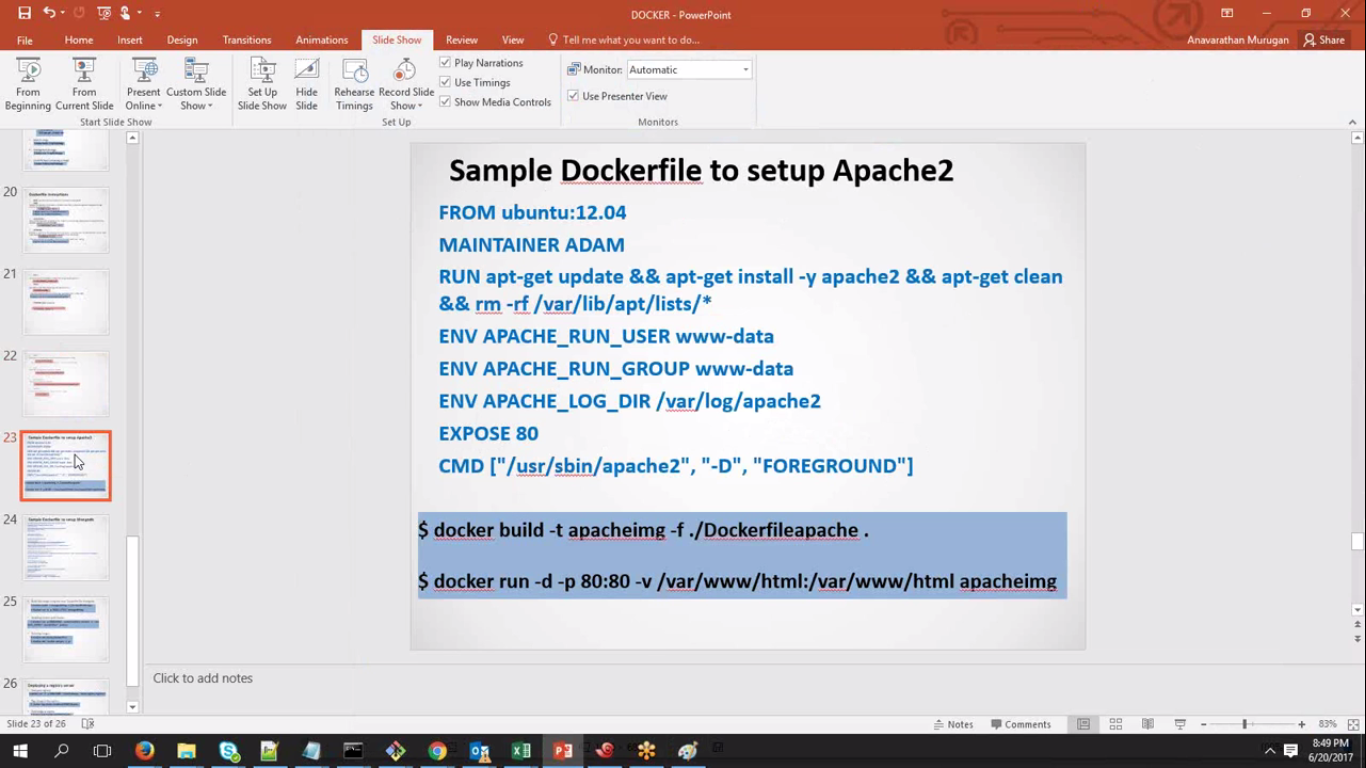
Ex: ADD test.tar.gz /var/www

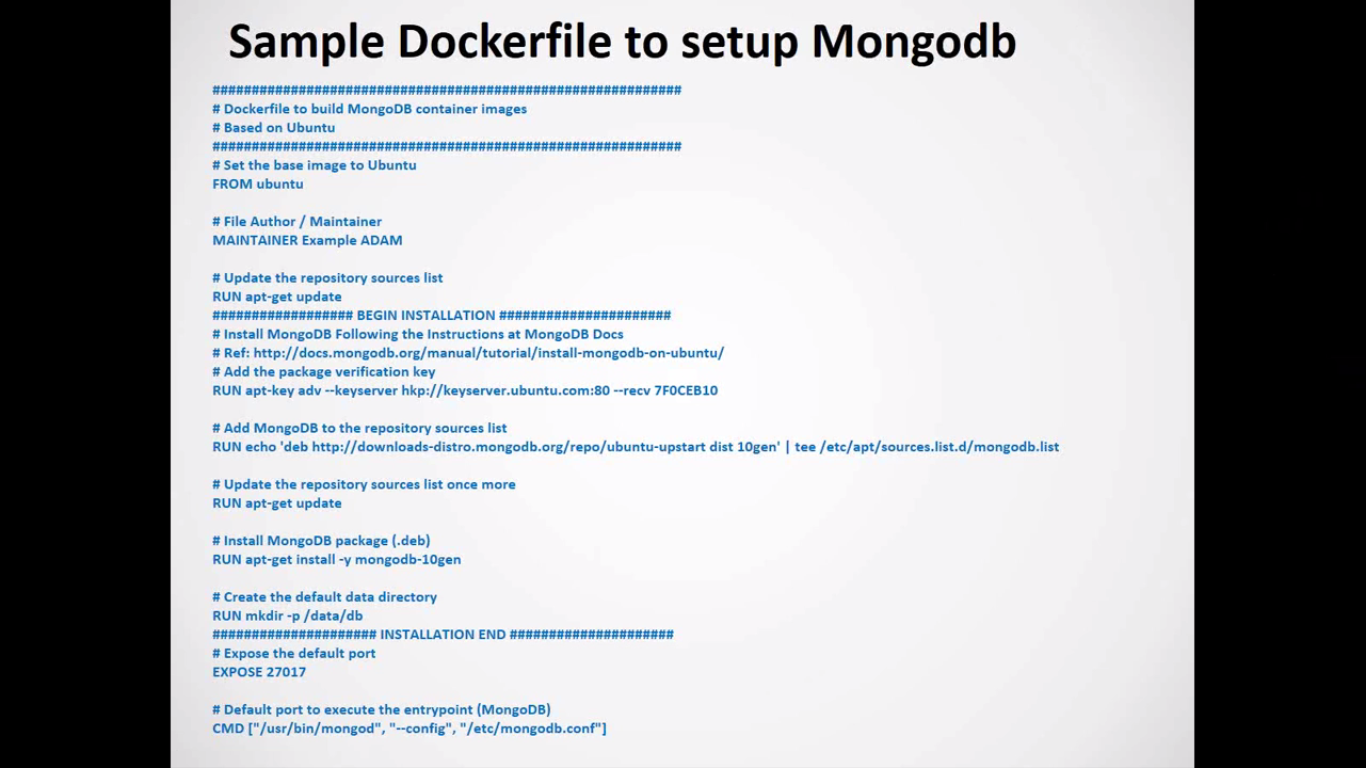
**MAINTAINER**: just for give the information about docker file author.

Ex: MAINTAINER srini y “srinumails@gmail.com”

**EXPOSE**: tell to docker what ports are to be published in this image

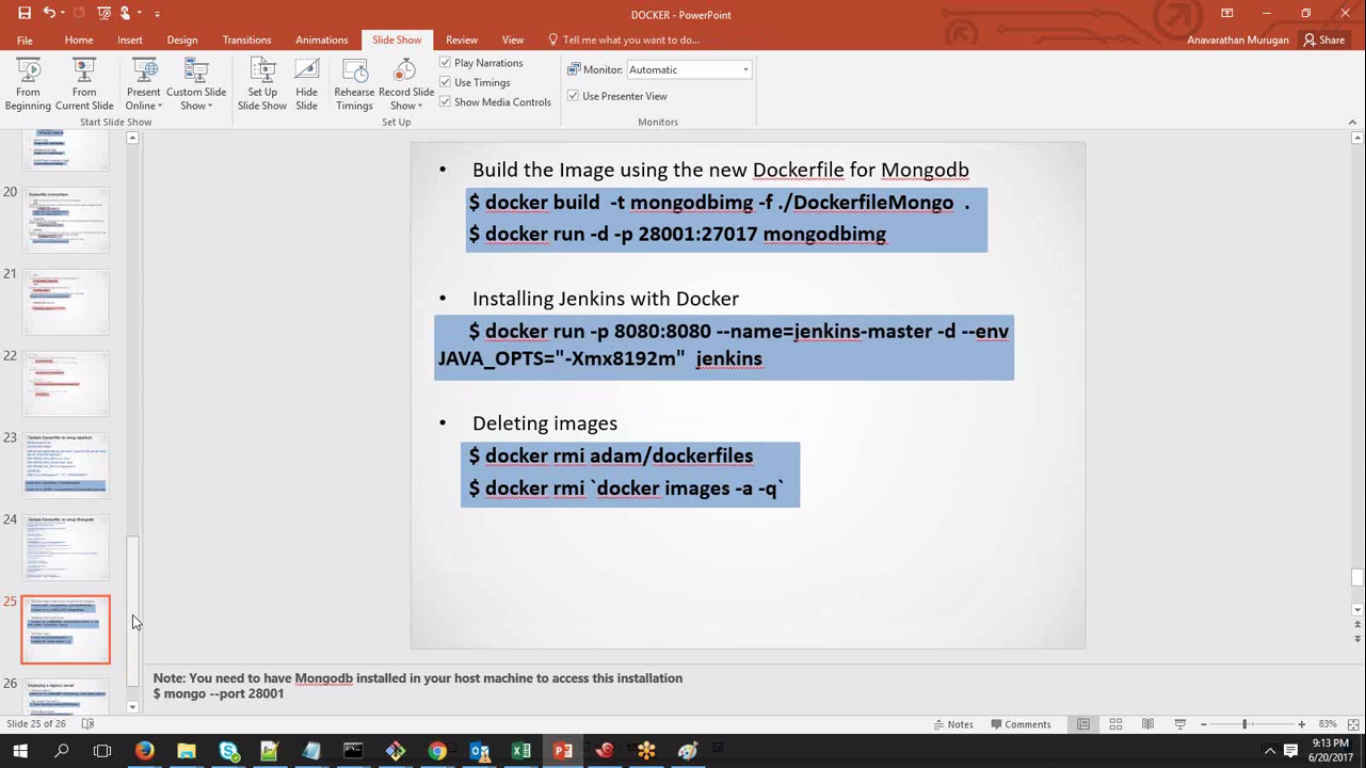
Ex: EXPOSE 8080





1. Install Jenkins using docker. 1) search for docker image using

$docker search jenkins



**Push images to docker Hub**

1. Create account in docker hub Ex: username: srinieisdev
2. Create repository in that. Ex: eisdcokertest
3. Tag your docker image (we can’t push our root images)

Ex: docker tag b02d6cf6c6af srinieisdev/eisdcokertest:firsttry

1. Push docker image into docker hub
2. Ex: docker push srinieisdev/eisdcokertest