Docker is a containerization tool.

Virtualization -- Fixed hardware allocation.

Containerization - No Fixed Hardware

Process isolation ( Dependency in os is removed )

+++++++++++++++++++++++++

In comparison to the traditional virtualization functionalities of hypervisors,

Docker containers eliminate the need for a separate guest operating system for every new virtual machine.

Docker implements a high-level API to provide lightweight containers that run processes in isolation.

A Docker container enables rapid deployment with minimum run-time requirements. It also ensures better management and simplified portability.

This helps developers and operations team in rapid deployment of an application.

+++++++++++++++++++++++++

Create Ubuntu Machine on AWS

All Traffic - anywhere

Connect using git bash

https://get.docker.com/

Go to Root Account

$ sudo su -

# curl -fsSL https://get.docker.com -o get-docker.sh ( this will download shell script in the machine)

# sh get-docker.sh ( This will execute the shell script, which will install docker )

How to check the docker is installed or not

# docker --version

We should be comformatable with four terms

1) Docker Images

Combinations of binaries / libraries which are necessary for one software application.

2) Docker Containers

When image is installed and in comes into running condition, it is called container.

3) Docker Host

Machine on which docker is installed, is called as Docker host.

4) Docker Client

Terminal used to run docker run commands ( Git bash )

On linux machine, git bash will work like docker client.

+++++++++++++++++++++++++++

Docker Commands

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

Working on Images

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

1 To download a docker image

docker pull image\_name

2 To see the list of docker images

docker image ls

(or)

docker images

3 To delete a docker image from docker host

docker rmi image\_name/image\_id

4) To upload a docker image into docker hub

docker push image\_name

5) To tag an image

docker tag image\_name ipaddress\_of\_local\_registry:5000/image\_name

6) To build an image from a customised container

docker commit container\_name/container\_id new\_image\_name

7) To create an image from docker file

docker build -t new\_image\_name

8) To search for a docker image

docker search image\_name

9) To delete all images that are not attached to containers

docker system prune -a

++++++++++++++++++++++++++++++++++++++++++++++

Working on containers

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

10) To see the list of all running continers

docker container ls

11) To see the list of running and stopped containers

docker ps -a

12) To start a container

docker start container\_name/container\_id

13) To stop a running container

docker stop container\_name/container\_id

14) To restart a running container

docker restart container\_name/container\_id

To restart after 10 seconds

docker restart -t 10 container\_name/container\_id

15) To delete a stopped container

docker rm container\_name/container\_id

16) To delete a running container

docker rm -f container\_name/container id

17) To stop all running containers

docker stop $(docker ps -aq)

18) To restart all containers

docker restart $(docker ps -aq)

19) To remove all stopped containers

docker rm $(docker ps -aq)

20) To remove all contianers(running and stopped)

docker rm -f $(docker ps -aq)

21) To see the logs generated by a container

docker logs container\_name/container\_id

22) To see the ports used by a container

docker port container\_name/container\_id

23) To get detailed info about a container

docker inspect container\_name/container\_id

24) To go into the shell of a running contianer which is moved into background

docker attach container\_name/container id

25) To execute anycommand in a container

docker exec -it container\_name/container\_id command

Eg: To launch the bash shell in a contianer

docker exec -it container\_name/container\_id bash

26) To create a container from a docker image ( imp )

docker run image\_name

++++++++++++++++++++++++++++++++++++++++++++++++

Run command options

-it for opening an interactive terminal in a container

--name Used for giving a name to a container

-d Used for running the container in detached mode as a background process

-e Used for passing environment varaibles to the container

-p Used for port mapping between port of container with the dockerhost port.

-P Used for automatic port mapping ie, it will map the internal port of the container

with some port on host machine.

This host port will be some number greater than 30000

-v Used for attaching a volume to the container

--volume-from Used for sharing volume between containers

--network Used to run the contianer on a specific network

--link Used for linking the container for creating a multi container architecture

--memory Used to specify the maximum amount of ram that the container can use

++++++++++++++++++++++++++++++++++++++++++++++++++=

# docker images ( There are no images )

To download tomcat image

# docker pull tomee

# docker images

# docker pull ubuntu

If you do not specify the version, by default, we get latest version

I want to download jenkins

# docker pull jenkins

TO create a container from an image

# docker run --name mytomcat -p 7070:8080 tomee

docker run --name c1 -p 7070:8080 tomee

TO check the tomcat is running or not

http://13.250.47.90:7070

( 7070 is port number mapped in docker host)

Lets remove the container ( Open another gitbash terminal)

# docker stop c1

# docker rm -f c1

# docker run --name mytomcat -p 7070:8080 -d tomee

( The above command runs tomcat in detached mode , so we get out # prompt back )

# docker container ls

TO start jenkins

# docker run --name myjenkins -p 9090:8080 -d jenkins

To check for jenkins ( Open browser )

http://13.250.47.90:9090

To create ubuntu container

# docker run --name myubuntu -it ubuntu

Observation: You have automatically entered into ubuntu

# ls ( To see the list of files in ubuntu )

# exit ( To comeout of container back to host )

+++++++++++++

Scenario 1:

Start tomcat as a container and name it as "webserver". Perform port mapping and run this container in detached mode

# docker run --name webserver -p 7070:8080 -d tomee

To access homepage of the tomcat container

Launch any browser

public\_ip\_of\_dockerhost:7070

++++++++++++++++++++++++++++++++

Scenario 2:

Start jenkins as a container in detached mode , name is as "devserver", perform port mapping

# docker run -d --name devserver -p 9090:8080 jenkins

To access home page of jenkins ( In browser)

public\_ip\_of\_dockerhost:9090

++++++++++++++++++++++++++++++++++++++++

Scenario 3: Start nginx as a container and name as "appserver", run this in detached mode , perform automatic port mapping

Generally we pull the image and run the image

Instead of pulling, i directly

# docker run --name appserver -P -d nginx

( if image is not available, it perform pull operation automatically )

( Capital P , will perform automatic port mapping )

How to check nginx is running or not? ( we do not know the port number)

To know the port that is reserved for nginx )

# docker port appserver

80/tcp -> 0.0.0.0:32768

80 is nginx port

32768 is dockerhost port

or

# docker container ls ( to see the port of nginz and docker host )

To check nginx on browser

52.221.192.237:32768

++++++++++++++++++++++++++

To start centos as container

# docker run --name mycentos -it centos

# exit ( To come back to dockerhost )

++++++++++++++

Scenario 3: Start nginx as a container and name as "appserver", run this in detached mode , perform automatic port mapping

Generally we pull the image and run the image

Instead of pulling, i directly

# docker run --name appserver -P -d nginx

( if image is not available, it perform pull operation automatically )

( Capital P , will perform automatic port mapping )

How to check nginx is running or not? ( we do not know the port number)

To know the port that is reserved for nginx )

# docker port appserver

80/tcp -> 0.0.0.0:32768

80 is nginx port

32768 is dockerhost port

or

# docker container ls ( to see the port of nginz and docker host )

To check nginx on browser

52.221.192.237:32768

++++++++++++++++++++++++++

To start centos as container

# docker run --name mycentos -it centos

# exit ( To come back to dockerhost )

++++++++++++++

To start mysql as container, open interactive terminal in it, create a sample table.

# docker run --name mydb -d -e MYSQL\_ROOT\_PASSWORD=sunil mysql:5

# docker container ls

I want to open bash terminal of mysql

# docker exec -it mydb bash

To connect to mysql database

# mysql -u root -p

enter the password, we get mysql prompt

TO see list of databases

> show databases;

TO switch to a databse

> use db\_name

> use mysql

TO create emp tables and dept tables

https://justinsomnia.org/2009/04/the-emp-and-dept-tables-for-mysql/

> exit

# exit

# exit

Multi container architecture using docker

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

This can be done in 2 ways

1) --link

2) docker-compose

1) --link option

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

Use case:

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

Start two busybox containers and create link between them

Create 1st busy box container

# docker run --name c10 -it busybox

/ #

How to come out of the container without exit

( ctrl + p + q)

Create 2nd busy box container and establish link to c1 container

# docker run --name c20 --link c10:c10-alias -it busybox ( c10-alias is alias name)

/ #

How to check link is established for not?

/ # ping c10

Ctrl +c ( to come out from ping )

( ctrl + p + q)

+++++++++++++++++++++++++++++++++

Ex 2: Creating development environment using docker

Start mysql as container and link it with wordpress container.

Developer should be able to create wordpress website

1) TO start mysql as container

# docker run --name mydb -d -e MYSQL\_ROOT\_PASSWORD=sunil mysql:5

( if container is already in use , remove it

# docker rm -f mydb )

Check whether the container is running or not

# docker container ls

2) TO start wordpress container

# docker run --name mysite -d -p 5050:80 --link mydb:mysql wordpress

Check wordpress installed or not

Open browser

public\_ip:5050

18.138.58.3:5050

++++++++++++++++++++++++++++++++++++++++++++++

Ex 3: Create LAMP Architecture using docker

L -- linux

A -- apache tomcat

M -- mysql

P -- php

( Linux os we already have )

Lets remove all the docker containers

# docker rm -f $(docker ps -aq)

# docker container ls ( we have no containers now )

1) TO start mysql as container

# docker run --name mydb -d -e MYSQL\_ROOT\_PASSWORD=sunil mysql:5

2) TO start tomcat as container

# docker run --name apache -d -p 6060:8080 --link mydb:mysql tomcat

TO see the list of containers

# docker container ls

To check if tomcat is linked with mysql

# docker inspect apache ( apache is the name of the container )

3) TO start php as container

# docker run --name php -d --link apache:tomcat --link mydb:mysql php

++++++++++++++++++++

ex 4:

Create CI-CD environment, where jenkins container is linked with two tomcat containers.

Lets delete all the container

# docker rm -f $(docker ps -aq)

To start jenkins as a container

# docker run --name devserver -d -p 7070:8080 jenkins/jenkins

to check jenkins is running or not

Open browser

public\_ip:7070

http://18.138.58.3:7070

We need two tomcat containers ( qa server and prod server )

# docker run --name qaserver -d -p 8080:8080 --link devserver:jenkins tomee

to check the tomcat use public\_ip but port number will be 8080

http://18.138.58.3:8080

# docker run --name prodserver -d -p 9090:8080 --link devserver:jenkins tomcat

to check the tomcat of prodserver

http://18.138.58.3:9090

+++++++++++++++++++++++++++++

Creating testing environment using docker

Create selenium hub container, and link it with two node containers.

One node with firefox installed, another node with chrome installed.

Tester should be able to run selenuim automation programs for testing the application on multiple browsers.

To delete all the running containers

#

In Browser -- open - hub.docker.com

Search for selenium

We have a image - selenium/hub

To start selenium/hub as container

# docker run --name hub -d -p 4444:4444 selenium/hub

In hub.docker.com

we also have- selenium/node-chrome-debug ( It is ubuntu container with chrome)

To start it as a container and link to hub ( previous container)

# docker run --name chrome -d -p 5901:5900 --link hub:selenium selenium/node-chrome-debug

In hub.docker.com

we also have- selenium/node-firefox-debug

To start it as a container and link to hub ( It is ubuntu container with firefox)

# docker run --name firefox -d -p 5902:5900 --link hub:selenium selenium/node-firefox-debug

To see the list of container

# docker container ls

Note: firefox and chrome containers are GUI containers.

To see the GUI interface to chrome / firefox container

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

Download and install vnc viewer

In VNC viewer search bar

public\_ip\_dockerhost:5901

18.136.211.65:5901

Password - secret

++++++++++++++++++++++++++++++++++++++++++++++++++++++

All the commands we learnt till date are adhoc commands.

In the previous usecase we have installed two containers ( chrome and firefox)

Lets say you need 80 containers?

Do we need to run 80 commands?

Instead of 80 commands, we can use docker compose

++++++++++++++++++++++++

Docker compose

This is a feature of docker using which we can create multicontainer architecture using yaml files. This yaml file contains information about the containers that we want to launch and how they have to be linked with each other.Yaml is a file format. It is not a scripting language.

Yaml will store the data in key value pairs

Lefthand side - Key

Righthand side - Value

Yaml file is space indented.

Sample Yaml file

---

CTELt:

SAI:

sunil: Devops

raj: Python

Coordinators:

lakshmi: Devops

rani: AWS

...

+++++++++++++++++++++++++

To validate the abvove Yaml file

Open http://www.yamllint.com/

Paste the above code -- Go button

Installing Docker compose

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

1) Open https://docs.docker.com/compose/install/

2) Go to linux section

Copy and pase the below two commands

# sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# sudo chmod +x /usr/local/bin/docker-compose

How to check docker compose is installed or not?

# docker-compose --version

+++++++++++++++++++++++++

Create a docker compose file for setting up dev environment.

mysql container is linked with wordpress container.

# vim docker-compose.yml ( Name of the file should be docker-compose.yml)

---

version: '3'

services:

mydb:

image: mysql:5

environment:

MYSQL\_ROOT\_PASSWORD: sunilsunil

mysite:

image: wordpress

ports:

- 5050:80

links:

- mydb:mysql

...

:wq

Lets remove all the running container

# docker rm -f $(docker ps -aq)

How to start the above services from dockerfile

# docker-compose up

We got lot of logs coming on the screen. to avoid it we use -d option

# docker-compose stop

Remove the container

# docker rm -f $(docker ps -aq)

# docker-compose up -d

To check wordpress

public\_ip:5050

++++++++++++++++++++++++++++

To stop both the containers

# docker-compose stop

++++++++++++++++++++++++++++++

Create a docker compose file for setting up LAMP architecture

# vim docker-compose.yml

---

version: '3'

services:

mydb:

image: mysql:5

environment:

MYSQL\_ROOT\_PASSWORD: sunilsunil

apache:

image: tomee

ports:

- 6060:8080

links:

- mydb:mysql

php:

image: php

links:

- mydb:mysql

- apache:tomcat

...

:wq

# docker-compose up -d

To see the list of the containers

# docker container ls

( Observation - we are unable to see the php container)

# docker ps -a

++++++++++++++++++++++++++++++++++++

Ex: Docker-compose file for setting up CI-CD Environment.

jenkins container is linked with two tomcat containers

# vim docker-compose.yml

---

version: '3'

services:

devserver:

image: jenkins/jenkins

ports:

- 7070:8080

qaserver:

image: tomee

ports:

- 8899:8080

links:

- devserver:jenkins

prodserver:

image: tomee

ports:

- 9090:8080

links:

- devserver:jenkins

...

:wq

# docker rm -f $(docker ps -aq)

# docker-compose up -d

# docker container ls

To check

public\_ip:7070 ( To check jenkins )

public\_ip:8899 ( Tomcat qa server )

public\_ip:9090 ( Tomcat prod server )

13.126.58.183:7070

13.126.58.183:8899

13.126.58.183:9090

+++++++++++++++++++++

Docker-compose file to set up testing environment.

selenium hub container is linked with two node containers.

# vim docker-compose.yml

---

version: '3'

services:

hub:

image: selenium/hub

ports:

- 4444:4444

chrome:

image: selenium/node-chrome-debug

ports:

- 5901:5900

links:

- hub:selenium

firefox:

image: selenium/node-firefox-debug

ports:

- 5902:5900

links:

- hub:selenium

...

:wq

Lets delete all the running containers

# docker rm -f $(docker ps -aq)

# docker-compose up -d

# docker container ls

As it is GUI container,

we can access using VNC viewer

Open VNC viewer

52.77.219.115:5901

password: secret

++++++++++++++++++++++++++++++++++++++++++++++

Docker volumes

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

Docker containers are ephemeral ( temporary )

Where as the data processed by the container should be permanent.

Generally, when a container is deleted all its data will be lost.

To preserve the data, even after deleting the container, we use volumes.

Volumes are of two types

1) Simple docker volumes

2) Docker volume containers ( Sharable volume )

Simple docker volumes

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

These volumes are used only when we want to access the data,

even after the container is deleted.

But this data cannot be shared with other containers.

usecase

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

1) Create a directory called /data ,

start centos as container and mount /data as volume.

Create files in mounted volume in centos container,

exit from the container and delete the container. Check if the files are still available.

Lets create a folder with the name

# mkdir /data

# docker run --name c1 -it -v /data centos ( v option is used to attach volume)

# ls ( Now, we can see the data folder also in the container)

# cd data

# touch file1 file2

# ls

# exit ( To come out of the container )

# docker inspect c1

We can see under mounts "data" folder it located in the host machine.

Copy the path

/var/lib/docker/volumes/c5c85f87fdc3b46b57bb15f2473786fe7d49250227d1e9dc537bc594db001fc6/\_data

Now, lets delete te container

# docker rm -f c1

After deleting the container, lets go to the location of the data folder

# cd /var/lib/docker/volumes/d867766f70722eaf8cba651bc1d64c60e9f49c5b1f1ebb9e781260f777f3c7e8/\_data

# ls ( we can see file1 file2 )

( Observe , the container is deleted but still the data is persistant )

+++++++++++++

docker volume containers

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

These are also known as reusable volume.

The volume used by one container can be shared with other containers.

Even if all the containers are deleted, data will still be available on the docker host.

Ex:

# sudo su -

Lets create a directory /data

# mkdir /data

Lets Start centos as container

# docker run --name c1 -it -v /data centos

# ls ( we can see the list of files and dir in centos )

# cd data

# ls ( currently we have no files )

Lets create some files

# touch file1 file2 ( These two files are available in c1 container)

Comeout of the container without exit

# Ctrl +p Ctrl +q ( container will still runs in background )

Lets Start another centos as container ( c2 container should use the same volume as c1)

# docker run --name c2 -it --volumes-from c1 centos

# cd data

# ls ( we can see the files created by c1 )

Lets create some more files

# touch file3 file4

# ls ( we see 4 files )

Comeout of the container without exit

# Ctrl +p Ctrl +q ( container will still runs in background )

Lets Start another centos as container

# docker run --name c3 -it --volumes-from c2 centos

# cd data

# ls ( we can see 4 files )

# touch file5 file6

# ls

Comeout of the container without exit

# Ctrl +p Ctrl +q ( container will still runs in background )

Now, lets connect to any container which is running in the background

# docker attach c1

# ls ( you can see all the files )

# exit

Identify the mount location

$ docker inspect c1

( search for the mount section )

Take a note of the source path

/var/lib/docker/volumes/e22a9b39372615727b964151b6c8108d6c02b13114a3fcce255df0cee7609e15/\_data

Lets remove all the container

# docker rm -f c1 c2 c3

Lets go to the source path

# cd /var/lib/docker/volumes/e22a9b39372615727b964151b6c8108d6c02b13114a3fcce255df0cee7609e15/\_data

# ls ( we can see all the files )

Docker volumes

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

Docker containers are ephemeral ( temporary )

Where as the data processed by the container should be permanent.

Generally, when a container is deleted all its data will be lost.

To preserve the data, even after deleting the container, we use volumes.

Volumes are of two types

1) Simple docker volumes

2) Docker volume containers ( Sharable volume )

Simple docker volumes

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

These volumes are used only when we want to access the data,

even after the container is deleted.

But this data cannot be shared with other containers.

usecase

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

1) Create a directory called /data ,

start centos as container and mount /data as volume.

Create files in mounted volume in centos container,

exit from the container and delete the container. Check if the files are still available.

Lets create a folder with the name

# mkdir /data

# docker run --name c1 -it -v /data centos ( v option is used to attach volume)

# ls ( Now, we can see the data folder also in the container)

# cd data

# touch file1 file2

# ls

# exit ( To come out of the container )

# docker inspect c1

We can see under mounts "data" folder it located in the host machine.

Copy the path

/var/lib/docker/volumes/08a4d35066c1f545a4c4233e03c26a9cdbceb453c39fbc187a6438424fae66c0/\_data

Now, lets delete te container

# docker rm -f c1

After deleting the container, lets go to the location of the data folder

# cd /var/lib/docker/volumes/08a4d35066c1f545a4c4233e03c26a9cdbceb453c39fbc187a6438424fae66c0/\_data

# ls ( we can see file1 file2 )

( Observe , the container is deleted but still the data is persistant )

+++++++++++++

docker volume containers

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

These are also known as reusable volume.

The volume used by one container can be shared with other containers.

Even if all the containers are deleted, data will still be available on the docker host.

Ex:

# sudo su -

Lets create a directory /data

# mkdir /data

Lets Start centos as container

# docker run --name c1 -it -v /data centos

# ls ( we can see the list of files and dir in centos )

# cd data

# ls ( currently we have no files )

Lets create some files

# touch file1 file2 ( These two files are available in c1 container)

Comeout of the container without exit

# Ctrl +p Ctrl +q ( container will still runs in background )

Lets Start another centos as container ( c2 container should use the same volume as c1)

# docker run --name c2 -it --volumes-from c1 centos

# cd data

# ls ( we can see the files created by c1 )

Lets create some more files

# touch file3 file4

# ls ( we see 4 files )

Comeout of the container without exit

# Ctrl +p Ctrl +q ( container will still runs in background )

Lets Start another centos as container

# docker run --name c3 -it --volumes-from c2 centos

# cd data

# ls ( we can see 4 files )

# touch file5 file6

# ls

Comeout of the container without exit

# Ctrl +p Ctrl +q ( container will still runs in background )

Now, lets connect to any container which is running in the background

# docker attach c1

# ls ( you can see all the files )

# exit

Identify the mount location

$ docker inspect c1

( search for the mount section )

Take a note of the source path

/var/lib/docker/volumes/4dcd29f66b45126f1aa739091c2933037868a5a3eadf41b1b4ca3ecabf5091c6/\_data

Lets remove all the container

# docker rm -f c1 c2 c3

Lets go to the source path

# cd /var/lib/docker/volumes/4dcd29f66b45126f1aa739091c2933037868a5a3eadf41b1b4ca3ecabf5091c6/\_data

# ls ( we can see all the files )

+++++++++++++++++++++++++++++++++++++++++++++++++++

Creating customized docker images

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

Whenever docker container is deleted,

all the softwares that we have installed within the container will also be deleted.

If we can save the container as an image, then we can preserve the softwares.

This creation of customized docker images can be done in two ways.

1) using docker commit command

2) using docker file

Using docker commit

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

# docker run --name c1 -it ubuntu

Update apt repository

# apt-get update

# apt-get install git

TO check the git

# git --version

# exit

TO save the container as image (snapshot )

# docker commit c1 myubuntu

To see the list of images

# docker images ( you can see the image which you have created )

++++++++++++++++++++++

Now lets run the image which we have created

# docker run --name c2 -it myubuntu

# git --version ( git is pre installed )

+++++++++++++++++++++++++++++++++++++++++++++++++

Using docker file

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

This is a simple text file, which uses predefinied keywords for creating customized docker images.

Key words used in docker file ( case sensitive )

1) FROM -- used to specify the base image from which the docker file has to be created.

2) MAINTAINER -- This represents name of the organization or the

author who created this docker file.

3) CMD -- This is used to specify the initial command that should be executed when the container starts.

4) ENTRYPOINT - used to specify the default process that should be executed when container starts.

It can also be used for accepting arguments from the CMD instruction.

5) RUN -- Used for running linux commands within the container. It is generally helpful for installing the software in the container.

6) USER -- used to specify the default user who should login into the container.

7) WORKDIR --

Used to specify default working directory in the container

8) COPY -- Copying the files from the host machine to the container.

9) ADD -- Used for copying files from host to container, it can also be used for downloading files from remote servers.

10) ENV -- used for specifying the environment variables that should be passed to the container.

EXPOSE -- Used to specify the internal port of the container

VOLUME -- used to specify the default volume that should be attached to the container.

LABEL -- used for giving label to the container

STOPSIGNAL -- Used to specify the key sequences that have to be passed in order to stop the container.

+++++++++++++++++++++

+++++++++++++++++++++++++++++++++

Create a dockerfile by taking nginx as the base image

and specify the maintainer as logiclabs. Construct an image from the dockerfile.

Creating customized docker images by using docker file.

$ sudo su -

# vim dockerfile

FROM nginx

MAINTAINER logiclabs

:wq

TO build an image from the dockerfile

# docker build -t mynginx .

( t stands for tag,

. stands for current working dir

mynginx is the new image name )

TO see the image

# docker images

++++++++++++++++++++++++++++++++++++++++++++++

When ever i start my container, i want a program to get executed.

# vim dockerfile

FROM centos

MAINTAINER logiclabs

CMD ["date"]

:wq

TO build an image from the dockerfile

# docker build -t mycentos .

TO see the image

# docker images

Running conainer from the image

# docker run -it mycentos

++++++++++++++++++++++++++

In one docker file, we can have one CMD instruction.

If we give two CMD instruction, it executes the latest one

Lets try

# vim dockerfile

FROM centos

MAINTAINER logiclabs

CMD ["date"]

CMD ["ls", "-la"]

:wq

# docker build -t mycentos .

# docker run -it mycentos

( Observation, we get ls -la output )

++++++++++++++++++++++++++++++++++++++++++++++

In ubuntu container, I want to install git in it.

Lets remove the docker file

# rm dockerfile

# vim dockerfile

FROM ubuntu

MAINTAINER logiclabs

RUN apt-get update

RUN apt-get install -y git

:wq

Note: CMD -- will run when container starts.

RUN -- will executed when image is created.

# docker build -t myubuntu .

Lets see the images list and space consumed by our image

# docker images

# docker run -it myubuntu

# git --version

# exit

++++++++++++++++++++++++++++++++++

Lets perform version controlling in docker file

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

# mkdir docker

# mv dockerfile docker

# cd docker

# ls

docker# git init

docker# git status

docker# git add .

docker# git commit -m "a"

( we get error we need to config git)

docker# git config --global user.name "sunildevops77"

docker# git config --global user.email "sunildevops77@gmail.com"

Now, run the above commit command ( git commit )

docker# vim dockerfile ( lets make some changes add another RUN command )

FROM ubuntu

MAINTAINER logiclabs

RUN apt-get update

RUN apt-get install -y git

RUN apt-get install -y default-jdk

:wq

docker# git add .

docker# git commit -m "b"

Now lets see the docker file

# vim dockerfile ( we see the latest one )

Now, I want to have previous version

# git log --oneline ( to see the list of all the commits)

We want to move to "a" commit ( take note of commit id )

# git reset --hard 10841c3

Now lets see the docker file

# vim dockerfile ( we see the old one )

+++++++++++++++++++++++++++++++++++

Cache busting

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

Whenever an image is build from a dockerfile, docker reads its memory and checks which instructions were already executed.

These steps will not be reexecuted.

It will execute only the latest instructions. This is a time saving mechanism provided by docker.

But, the disadvantage is, we can end up installing software packages from a repository which is updated long time back.

Ex:

# cd docker

# vim dockerfile

Lets just add one more instruction

FROM ubuntu

MAINTAINER logiclabs

RUN apt-get update

RUN apt-get install -y git

RUN apt-get install -y tree

:wq

Lets build an image

# docker build -t myubuntu .

( Observe the output, Step 2, 3, 4 is using cache. Only step 5 is executed freshly )

Advantage: time saving mechanism

Disadvantage : Lets say, you are running after 4 months, We are installing tree from apt which is updated long time back. )

TO avoid this disadvanatge we use cache busting

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

Note: cache busting is implemented using && symbol.

Which ever statement in the docker file has && will be re-executed.

# vim dockerfile

FROM ubuntu

MAINTAINER logiclabs

RUN apt-get update && apt-get install -y git tree

:wq

Lets build an image

# docker build -t myubuntu .

( Observe the output, step 3 - It is not using cache )

Ex: Create a dockerfile, for using ubuntu as base image, and install java in it.

Download jenkins.war and make execution of "java -jar jenkins.war" as the default process.

Every docker image come with default process.

As long as default process is running, the container will be running condition.

The moment, the default process is closed, the container will be exited.

Lets remove all the container

# docker rm -f $(docker ps -aq)

Observation 1:

When we start ubuntu container, we use below command

# docker run --name c1 -it ubuntu

/#

To comeout of the container we use Ctrl + p + q

# docker container ls

( our container c1 is running in the background )

Observation 2:

When we start jenkins container, we use below command

# docker run --name j1 -d -P jenkins/jenkins

Now, I want to open interactive terminal to enter jenkins

# docker exec -it j1 bash

( In ubuntu container, I can directly go into -it terminal,

where as in jenkins i am running an additional command exec ? )

Lets try to go to interactive terminal in docker run command )

# docker run --name j2 -it jenkins/jenkins

( we are not getting interactive terminal )

I want to run tomcat as container

# docker run --name t1 -d -P tomee

Lets find the reason

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

docker container ls ( to see the list of containers )

Observer the command section.

It tells you the default process that gets executed, when we start the container.

Container Default process

tomcat catalina.sh

jenkins /bin/tini

ubuntu /bin/bash

bash -- is nothing but the terminal.

For linux based container, the default process is shell process

( ex of shell process are bash shell, bourne shell etc )

Hence we are able to enter -it mode in ubuntu )

We are trying to change the default process of the container.

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

# vim dockerfile

FROM ubuntu

MAINTAINER logiclabs

RUN apt-get update

RUN apt-get install -y default-jdk

ADD http://mirrors.jenkins.io/war-stable/latest/jenkins.war /

ENTRYPOINT ["java","-jar","jenkins.war"]

:wq

Build an image from the dockerfile

# docker build -t myubuntu .

TO see the list of images ( we can see our new image )

# docker image ls

TO start container from new image

# docker run myubuntu ( Observe the logs generated on the

screen, we got logs related to jenkins , jenkins is fully up and running )

Its an ubuntu container, it is behaving as a jenkins container )

Ctrl +c

RUn the below command

# docker ps -a

For myubuntu the command is java -jar jenkins.war

For ubuntu the commans is /bin/bash

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Working on docker registry

Registry is a location where docker images are saved.

Types of registry

1) public registry

2) private registry

public registry is hub.docker.com

Images uploaded here are available for everyone.

Usecase: Create a customized ubuntu image, by installing tree in it.

Save this container as an image, and upload this image in docker hub.

Step 1: Create a new account in hub.docker.com

Step 2: Creating our own container

# docker run --name c5 -it ubuntu

Lets install tree package in this container

/# apt-get update

/# apt-get install tree

/# exit

Step 3: Save the above container as an image

# docker commit c5 sunildevops77/ubuntu\_img291

( sunildevops77/ubuntu\_img291 -- is the image name )

Note: Image name should start with docker\_id/

To see the list of images

# docker image ls ( we can see the new image )

TO upload the image to hub.docker.com ( docker login command is used )

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

# docker login ( provide docker\_id and password )

To upload the image

# docker push <image\_name>

# docker push sunildevops77/ubuntu\_img291

login to docker hub to see your image

+++++++++++++++++++++++++++++++++++++++++++++++++++

Container orchestration

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

This is the process of running docker containers in a distributed environment, on multiple docker host machines.

All these containers can have a single service running on them and they share the resources between eachother, even running on different host machines.

Docker swarm is the tool used for performing container orchestration

Advantages

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

1) Load balancing

2) scaling of containers

3) performing rolling updates

4) handling failover scenarios

+++++++++++++++++++++++++++++

Machine on which docker swarm is installed is called as manager.

Other machines are called as workers.

Lets create 3 machines

Name is as Manager, Worker1, Worker2

All the above machines should have docker installed in it.

Install docker using get.docker.com

( Optional step to change the prompt )

After installing docker in the 1st machine ( Manager ), Lets change the host name.

Host name will be available in the file hostname. We will change the hostname to manager.

# vim /etc/hostname

Manager

:wq

After changing the hostname, lets restart the machine

# init 6

+++++++++++++++++++++++++++

Similary repeat the same in worker1 and worker2

++++++++++++++++++++++++++++++++++++

Connect to Manager, install docker swarm in it.

$ sudo su -

Command to install docker swarm in manager machine

# docker swarm init --advertise-addr private\_ip\_of\_manager

# docker swarm init --advertise-addr 172.31.27.151

Please read the log messages

Now, we need to add workers to manager

Copy the docker swarm join command in the log and run in the worker1 and worker2

Open another gitbash terminal, connect to worker1

sudo su -

# docker swarm join --token SWMTKN-1-0etsmfa26vreeytq278q8ohhi73il7j1lpnrzzlowuld1r8yex-9x04pjmiq85jxjzjayzlglh1c 172.31.27.151:2377

Repeat for worker2

+++++++++++++++++++++++++++++

TO see the no of nodes from the manager

Manager # docker node ls ( we can see manager, worker1 and worker 2)