<https://learndocker.online/>

**History**

Initially One Physical Server was used for each application. There was a waste of resources. Then Hypervisor concept come in picture where one physical server but on top it multiple Virtual machine were created. It’s optimized the resource utilization but each VM have own OS which need to be updated and starting VM was slow, so Docker concept come into picture.

Docker remove the Hypervisor, now OS will on top of Hardware and OS and then Docker, so we have one OS and will take care of all application.

**What is Container**

Container is same like VM. It contain all information to run application but without OS

Container run on top of Docker.

Container are portable, we can use it anywhere

**Docker Installation**

Download from docker website and drag and drop in application

sudo apt-get update

sudo apt-get install -y docker.io

**Docker Installation on Old Version of OS**

Download toolbox from docker website for OS 🡪 Toolbox will install automatically docker machine 🡪 we can manage it using docker-machine command 🡪 to list all the docker run docker-machine ls command 🡪 client to communicate with docker server 🡪 run command docker-machine env default 🡪 run eval $(docker-machine env default) 🡪 then run docker-machine active 🡪 run docker-machine version

To create additional docker-machine 🡪 run command docker-machine create --driver vmwarefusion dockermachine\_name 🡪

**Checking Docker Service is running**

sudo systemctl status docker

sudo systemctl start docker && sudo systemctl enable docker

**Creating Docker Image, Run Container and Connecting to Container**

We need four command for that

create Create container from image

ps check the run containers

start start container

attach attach the I/O

rm remove the container

**Docker Images**

Docker Container is based on Docker Images. It’s a template for container. From one

image we can create multiple containers.

**docker images**

above command will show all the images on local server

**GETTING INFO**

**docker info**

above command will show all the information about all our container which we have on

our server**.**

**CREATION CONTAINER (1)**

**docker run hello-world**

**docker run -it hello-world bash**

**docker run -it alpine sh**

above command will download images if not present local and connect to it and run the container for it.

**docker create -it ubunt:16.04 bash**

above command will download the ubuntu image to your system

**docker run –name webserver -v $(pwd):/usr/share/nginx/html -d -p 8080:80 nginx**

**docker run -it ubuntu bash**

**docker run hello-world testing image**

docker run basic command

--name give any name to image

-v mapping local directory to container directory

-d run container in detached mode

-p mapping port of local machine to container

here 8080 is representing local and 80 is for container

local:container

Nginx image name from docker hub

**CREATING IMAGE WITH SPECIFIC VERSION (1a)**

**docker image ls**

above command will show list of images on local system

**docker image pull ubuntu:latest**

**docker image pull ubuntu:tag**

**docker image pull ubuntu:20.04**

above command will pull latest ubuntu images to local system

**docker image rm image\_name:tag**

above command will delete image with tag specified, if no tag , mean latest

**CREATING IMAGE USING DOCKER FILE (1b)**

Create a file with name **Dockerfile** and Enter following command

**FROM alpine:latest # from base image**

**RUN apk update # running cammand**

**RUN apk add bash # running command**

**CMD bash # give instruction to image**

Save above file and then run below command to create image

**docker image build myalpine:latest directory\_path**

to start the container for image run the below command

**docker container run -it myalpine:latest**

**PUSHING OUR IMAGES TO DOCKER HUB OUR REPO (1c)**

First tag image with personal repository with below command

**docker image tag myalpine:latest repository\_name/myalpine:latest**

**docker push repository\_name/myalpine:latest**

above command will make available our image on docker hub

**CHECKING RUNING OR EXISTING CONTAINERS (2a)**

**docker ps -a**

**docker container ls**

**docker container ls -a**

above both command will list all downloaded images in system

**GETTING HELP**

**docker --help**

above command will show all command for help

**docker run –help**

above command will give help of run command only

**STARTING CONTAINER OR CONNECTING TO EXISTING CONTAINER (2b)**

After run the container if we **press Ctlr+D**, this will terminate the container.

If we press **ctrl+p and Ctrl+q,** This will close the container in running state

**docker container run -it --name c1**

above command will run container with name c1, and put in interactive mode, to keep

container running press Ctrl+P and Ctrl+Q

**docker container run -itd --name c1**

above command will run container with name c1 in de-tached mode, and put in interactive mode

**docker container attach c1**

above command is used to connect to container again

**docker container exec container\_name linux\_command**

**docker container exec c1 cat /etc/nginx/confd**

above command will display contain of conf file on screen if the container is running

**docker container start container\_id**

above command will start the stop / exited container

**docker container stop container\_id**

above command will stop container

**docker container kill container\_id**

above command will stop container

**docker container log -f contaniner\_name**

if the container is running in detached mode and we will run above command will

display log of the container

**LOGIN TO DOCKER / CONNECTING TO DOCKER**

**docker start container\_id**

**docker attach container\_id**

above command will start container and connect to container

**LOGOUT FROM CONTAINER**

exit

above command used to come out from container

**DELETING CONTAINER (2a)**

**docker rm container\_id**

above command is used to delete the container one at a time

**docker rm $(docker container ls -aq) or docker rm $(docker -ps -aq)**

above command will delete all stop containers

**docker kill contanair\_id**

above command will stop the container

**docker run –rm -it image\_name**

above command will create container and delete automatically when exit

**DELETING IMAGES (2c)**

**docker rmi image\_id**

above command is used to delete the image

to delete the image, make sure there is not container running with this image.

**docker rmi image\_id –force**

above command will delete image force fully

**docker rmi $(docker images -aq ) -force**

above command will delete all image force fully

**MAPING LOCAL DIRECTORY TO DOCKER IMAGE AT CREATION TIME**

**docker create -it -v $(pwd): /var/www/ ubuntu:16.04 bash**

above command will map current working directory to container /var/www directory

**STARTING AND CREATING IMAGE AT SAME TIME**

**docker run -it -d ubuntu:16.04 bash**

above command combine the start and create command

**STOPING THE CONTAINER**

docker stop image\_id

above command will stop the container

**CREATING NGINX WEB SERVER USING DOCKER CONTAINER (3)**

Go to hub.docker.com and search for nginx 🡪 run command docker container run -p 80:80 nginx 🡪 checking now using browser 🡪 congrates

**USING VOLUME IN CONTAINER (3a)**

To Map local system directory to Container directory, we use volume tag

**docker run -p 80:80 -v local\_directory\_path:container\_directory\_path nginx**

**CONTAINER COMMINCATION WITH EACH OTHERS**

If we have two container, then they can ping each other by IP address

If we want to ping by container name then, run the container with following command

**docker container run -it --name c2 --link c1 alpine sh**

above command will run alpine OS container with name c2 and establish a link with

already running container name c1

**CREATING NETWORK**

In docker use default network with name bridge.

**docker container network ls**

above command will list all the available network

**docker network create network\_name**

above command will create user define network

to add the container to user define network add **the --network network\_name** tag to

run the container. If not network tag is specified default bridge network is used.

**docker container run --name c2 -it --network test alpine sh**

**CREATING A DOCKERFILE**

create a file with name Dockerfile without any extension and write following code

# Dockerfile

FROM nginx:alpine # this specify the image name

VOLUME /usr/share/nginx/html #map container directory

EXPOSE 80 #map the port 80 of container

Now runt the following command from Dockerfile directory

docker build . -t webserver:v1

the . is specify the Dockerfile location

-t is used to tagged the image with name webserver:v1

With this we are not pulling image from docker hub we create our own images

To see all images, run below command

docker images

now run below command to build the image and run the container

docker run -v $(pwd):/usr/share/nginx/html -d -p 8080:80 webserver:v1

to automate the above command, then we need docker compose

install docker-compose from docker website

check docker-compose version

docker-compose –verison

now create another file with name docker-compose.yml and enter following code

# docker-compose.yml

version: ‘2’

services:

webserver:

build: .

ports:

* “8080:80”

volumes:

* .:/usr/share/nginx/html

Save file and run below command

Docker-compose up (-d)

-d means in detached mode

docker-comose ps

docker-compose stop

docker commit container\_id

above command will save the container as a new image

docker tag container\_id any\_name

above command will give name to container

docker logs container\_id

above command will show what’s going wrong with container

**ctrl + p then ctrl + q** command is used to exit from docker attach command keep the container still running state

if the container is running in one windows, and we want another session to it then run

command

docker exec -ti container\_id bash

above command will connect to same container, if one session end, another also end

**DOCKER FLOW**

Docker Images 🡪 Container 🡪 Stop Container 🡪 Commit Container

**NETWORKING BETWEEN CONTAINERS**

**VOLUMES / SHARING DATA BETWEEN CONTAINER AND HOST & CONTAINERS**

Volumes are shared folder. Used to share data between container and host and

containers.

**Two types of Volumes are there**

1. Permanent if container delete, volumes on host still exist
2. Ephemeral if container delete, volumes also delete

**Note:**

Volumes are not part of images

**Sharing Database between Host & Container**

Create a folder using mkdir command

Then spin a container using below command to share the volume

**docker run -ti -v /path\_of\_directory\_on\_host:/path\_on\_container ubuntu bash**

**Sharing Database between Containers**

**docker run -ti –name server1 -v /share\_data ubuntu bash**

above command create volume with name share\_data but not sharing with host

**docker run -ti --name server2 --volumes-from server1 ubuntu bash**

above command will share the volumes of server1 with server2 container

**Note:**

Volumes shared between container is example of ephemeral

DOCKER REGISTERIES

docker repository are location from where we can download or pull docker

images.

Searching Images in docker repository

docker search image\_name

Login to docker from command line

docker login

docker pull ubuntu

docker push image\_id

above command will push the image to dockerhub.io

DOCKER FILES

Dockerfile is a small program to create image

**docker build -t name-of-image .**

. represent the location of Dockerfile

FROM image\_name

MAINTNER Rizwan khan <rizwan@gmail.com>

RUN echo “any command need to run on container”

CMD echo “hello container”

ADD used to add a local file

ADD run.sh /run.sh

ENV ENV DB\_HOST=db.production.example.com

ENV DB\_PORT=1234

ENTRYPOINT

EXPOSE EXPOSE 8080 use for port mapping

VOLUME VOLUME [‘/host/path/’ “/container/path/”]

VOLUME [“/shared-data”]

WORKDIR /install/

USER Rizwan

<https://www.docs.docker.com/engine/reference/builder/>

for more commands

Save above file with name Dockerfile and run below command

**docker build -t hello .**