<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.

**Docker Installation**

Download from docker website store.docker.com 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 vmware fusion dockermachine\_name 🡪

**Checking Docker Service is running**

sudo systemctl status docker

sudo systemctl start docker && sudo systemctl enable docker

**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

**GETTING INFO**

**docker info**

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

our server**.**

**DOCKER IMAGES (1)**

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

image we can create multiple containers.

**docker images ls**

above command will show all the images on local server

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

**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

**DELETING IMAGES (1b)**

**docker image rm image\_id:tag**

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

**CREATING OWN IMAGE USING DOCKER FILE (1c)**

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

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

**RUN apk update # running command**

**RUN apk add bash # running command**

**CMD bash # default command executed on container**

**COPY localfile\_path container\_location\_path**

**Copy will copy the along the image create to container**

**ADD #same as Copy command but extract and**

**download feature will provide**

**EXPOSE 90 # This command is used to publish port on container**

Save above file and then run below command to create image

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

to start the container based on our own image run the below command

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

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

First tag image with personal repository with below command

**docker image tag image\_name:tag docker\_id\_name/image\_name:tag**

**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

**COMMENTS IN DOCKERFILE (1e)**

# Sign is used for comments. Comment sign should always be at beginning of line,

otherwise docker will consider as arguments

**CREATION CONTAINER (2)**

**docker container run hello-world**

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

**docker container run -it alpine sh**

**docker container run -it ubuntu bash**

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

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

above command will download the ubuntu image of tag 16.04 to your system

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

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

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

**docker ps -a**

**docker container ls**

**docker container ls -a**

above both command will list all running or existed containers in system

**GETTING HELP**

**docker --help**

above command will show all command for help

**docker container 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, if the container is in running state

if the container is in existed state, then first start the container with below command and then run

above attach command to connect to it.

**docker container start container\_id or name**

above command will start the stop / exited container

**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 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

**CONNECTING TO DOCKER EXISTED CONTAINER (2c)**

**docker container start container\_id**

**docker container 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 (2d)**

**Docker container rm container\_id**

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

**docker container 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

**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 container stop container\_id

above command will stop the container

**CREATING NGINX WEB SERVER USING DOCKER CONTAINER (2e)**

Go to hub.docker.com and search for nginx 🡪 run command

**docker container run -p 80:80 nginx 🡪 checking now using browser 🡪 congrates**

**PERSISTING DATA**

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

To Map local system directory to Container directory, we use volume tag, it is called bind mount

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

**docker run -p 80:80 --mount type=bind, src=local\_directory, destincation=container\_directory, readyonly**

**nginx:latest**

above command with --mount and --volume are same but syntax is different. It is called bind mount

**MOUNTING THE VOLUME IN CONTAINER**

**docker container run -it --volume /data nginx:latest**

above command will run the container and create anonymous volume as well

**docker container run -it --volume my\_volume:/data nginx:latest**

above command will mount data volume with name my\_volume in container

**docker volume ls**

above command will list all the volume in container

**if we delete the container the volume will not be deleted**

**docker container run -it --mount type:volume, source:my\_volume, destination:/data nginx**

above command & the first command are same but only syntax difference, this command is used to create volume using --mount option

**DIFFERENCE BETWEEN MOUNT AND VOLUME**

The main difference that the volume is take care by docker and mount is care by host file system

**CREATING DOCKER VOLUME**

**docker create volume volume\_name**

above command will create a volume

**DELETING VOLUME**

**docker volume rm volume\_id or volume\_name**

above commad will delete the volume

**docker volume inspect volume\_id or volume\_name**

above command will give info about volume

**docker container inspect container\_id**

above command will give information about container

**docker volume prune**

above command will delete all unused volume in one

MOUNTING VOLUME FROM ANOTHER CONTAINER

docker container run -it –rm

**NETWORKING**

**CONTAINER COMMINCATION WITH EACH OTHERS USING LINKS**

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 c1 alpine sh**

**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. Now we can ping using name & IP

**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 c1 -it --network test alpine sh**

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

now we can ping c1 and c2 alternately.

**DOCKER COMPOSE**

A configuration file for docker compose are written in YAML language. Below is sample yaml file

name: value

name: Rizwan

user:

username: Rizwan

attributes:

country: Pakistan

skills: docker

language:

* Python
* Ruby

List of item are displayed with – (dash)

Save the file with .yaml extension

**CREATING A COMPOSE FILE**

In this compose file we will create nginx container, publish port 80 and publish customer page

Create a file docker-compose.yml 🡪

version: ‘3.3’

services:

web:

image: nginx:latest

ports:

- 80:80

Volumes:

- ./html : /usr/share/nginx/html

pg:

image: postgresql:9.6-alpine

# environment:

# - POSTGRES\_DB=TEST

env\_file: # declare environment var in file

- . /db.env

volumes:

- pg-data: /var/lib/postgresql/data # create name volume part 1

volumes: # create name volume part 2

pg-data

alpine: # start another container

image: alpine:latest

stdin\_open: true # equal to -i command

tty: true # equal to -t command

command: sh # bash shell

Create a file with name db.env and enter following information

POSTGRES\_DB=web\_app\_db

POSTGRES\_USER=app

POSTGRES\_PASSWORD=secret

Here services command show container, web is the name of container, image , port and volume

are self explanatory

now use below command to spin the container

**docker-compose up # all container will run**

**docker-compose up pg # only container with name pg will run**

**docker-compose up -d pg # only container with name pg will run in detached mode**

**docker-compose up -d # run all container in deteached mode**

**docker-compose ps # show all running containers**

above command is used to spin the container

**docker-compose down**

above command is used to delete all the recourses consumed by docker-compose up

**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**

**PUBLISHING PORTS**

**docker container run -p 123:567 nginx**

above command will map port no. 123 on local host with port no. 567 on

docker container

**docker container run -it -p 192.168.3.1:123:567 nginx**

above command is same but map ip is well

**docker container run -it -p 80:80 -p 81:81 nginx**

**docker container run -it -p 80-90:80-90 nginx**

above command is used to map multiple port

**LINKS**

We are using --link tag to connect two container to talk to each other

docker container run -it --link container\_name ubuntu:latest

**USER DEFINE NETWORKS**

**docker network ls**

above command is used to list available networks

**docker network create network\_name**

above command will create user define network

**docker network inspect network\_name**

above use to inspect the network

**docker network rm network\_name**

above command is used to delete the network

**--network network\_name tag is to connect any container**

**--ip ip\_address tag is used to assign ip address to container**

**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 .**

Setup Magento 2 in Docker

1. Setup Nginx Server as a container

Run below command to download the nginx docker image to your local system.

docker image pull nginx