**DOCKER**

**Docker Setup**

* **Launch EC2 instance**
  1. **Give any name like 🡪 Docker Server**
  2. **Use Ubuntu 18.0 Image**
  3. **Use Instance type** 🡪 **t2.micro**
  4. **Create security group**
  5. **Allow all traffic from anywhere**
  6. **Create a key pair**
  7. **Launch EC2 instance**
* **Go to** [**https://get.docker.com/**](https://get.docker.com/)
* **Use below commands to quick install Docker for Ubuntu machines**
* $ curl -fsSL https://get.docker.com -o get-docker.sh
* $ sh get-docker.sh
* **Give permissions for user “Ubuntu” to execute Docker commands**
  + You can check the Docker group 🡪 sudo vim /etc/group
  + You can give user “ubuntu” for docker group or you can use usermod command 🡪 **sudo usermod -aG docker ubuntu**
  + Check it 🡪 $ **id ubuntu**
  + Now you can run the docker commands

**Docker Image**

**A Docker image is a combination of bin/libs that are necessary for a software application to work. Initially all the software’s of docker are available in the form of docker images.**

**You can refer the Docker repository where you will find all kinds of docker images**

[**https://hub.docker.com/**](https://hub.docker.com/)

**Docker Container**

**A running instance of an image is called as a docker container**

**Docker Host**

**The server where docker is installed is called docker host**

**Docker Client**

**This is the CLI of docker where the user can execute the docker commands. The docker client accepts these commands and passes them to a background process called "docker daemon"**

**Docker daemon**

**This process accepts the commands coming from the docker client and routes them to work on docker images or containers or the docker registry**

**Docker Registries**

* **Storage for Docker Images**
* **Dockerhub is a default Registry**
* **Cloud based Registries**
  + - **DockerHub**
    - **GCR (Google Container Registry)**
    - **ACR (Azure Container Registry)**
    - **Amazon ECR (Elastic Container Registry)**
* **InHouse or Local Registries**
  + - **Nexus 3+**
    - **Jfrog Artifactory**
    - **DTR (Docker Trusted Registry)**

**Docker Importance Commands**

1. **$ docker images 🡪 List all the images locally**
2. **$ docker run 🡪 Creates new container**
3. **$ docker ps 🡪 List running containers**
4. **$ docker ps -a 🡪 List all the containers**
5. **$ docker exec 🡪 Execute commands on containers**
6. **$ docker start/stop/restart container\_id / container\_name**
7. **$ docker rmi image\_name / image\_id 🡪 Remove docker images locally**
8. **$ docker rmi -f image\_name / image\_id 🡪 Forcibly remove the docker image associated with a running container**
9. **$ docker inspect (docker image inspect image\_name) 🡪 Detail of Container & Image**
10. **$ docker pull 🡪 Pull the docker images from Docker Repository**
11. **$ docker system prune -af 🡪 To delete all images**
12. **$ docker rm container\_name / container\_id 🡪 To remove the container**
13. **$ docker rm -f container\_name / container\_id 🡪 To remove forcibly a running container**
14. **$ docker stop $(docker ps -aq) 🡪 To stop all the running containers**
15. **$ docker rm $(docker ps -aq) 🡪 To delete all the stopped containers**
16. **$ docker rm -f $(docker ps -aq) 🡪 To delete all the running containers**
17. **ctrl+p,ctrl+q 🡪 To come outside of the container safely**

**Examples:**

1. $ docker run –name myweb nginx
2. $ docker run –name myweb -d nginx
3. $ docker run –name myweb -d -p 8080:80
   1. Here 8080 🡪 is host port
   2. And 80 🡪 is container port
4. $ docker stop container\_name / container\_id
5. $ docker start containr\_name / container\_id
6. $ docker exec container\_name ls /
7. $ docker exec -it container\_name /bin/bash
   1. We can run commands inside container
   2. Run “ps” command
   3. You can install it “apt update” 🡪 “apt install procps -y”
   4. Run “ps -ef”
8. $ docker rmi nginx:tag\_name
9. $ docker run -it ubuntu /bin/bash

**Bind Mount:**

**Bind mount is mostly used inject data from host machine to container. But for preserving the data the best option is docker volumes.**

**Example:**

* 1. $mkdir mysql\_backup
  2. $ docker run –name mydb -d -e MYSQL\_ROOT\_PASSWORD=Password123 -p 3030:3306 -v /home/ubuntu/mysql\_backup:/var/lib/mysql mysql
  3. verify it 🡪 $ docker exec -it mydb /bin/bash
  4. $ cd /var/lib/mysql
  5. $ ls

**Docker Volumes**

**Containers are ephemeral or you can say like containers are volatile in nature. Means if the container crashes or exited it will lose the data inside it. In order to save the data inside the container we will use docker volumes.**

**Command:**

* + 1. **Docker volume**
    2. **Docker volume create mydbdata**
    3. **Docker run –name mydb -d -p 3030:3306 -e MYSQL\_ROOT\_PASSWORD=password -v mydbdata:/var/lib/mysql mysql**
    4. **Sudo su –**
    5. **Ls /var/lib/docker/volumes/ 🡪 you should able to see your docker volume there**
    6. **You can check the volume bonded to this container using docker inspect command**

**$ docker inspect mydb**

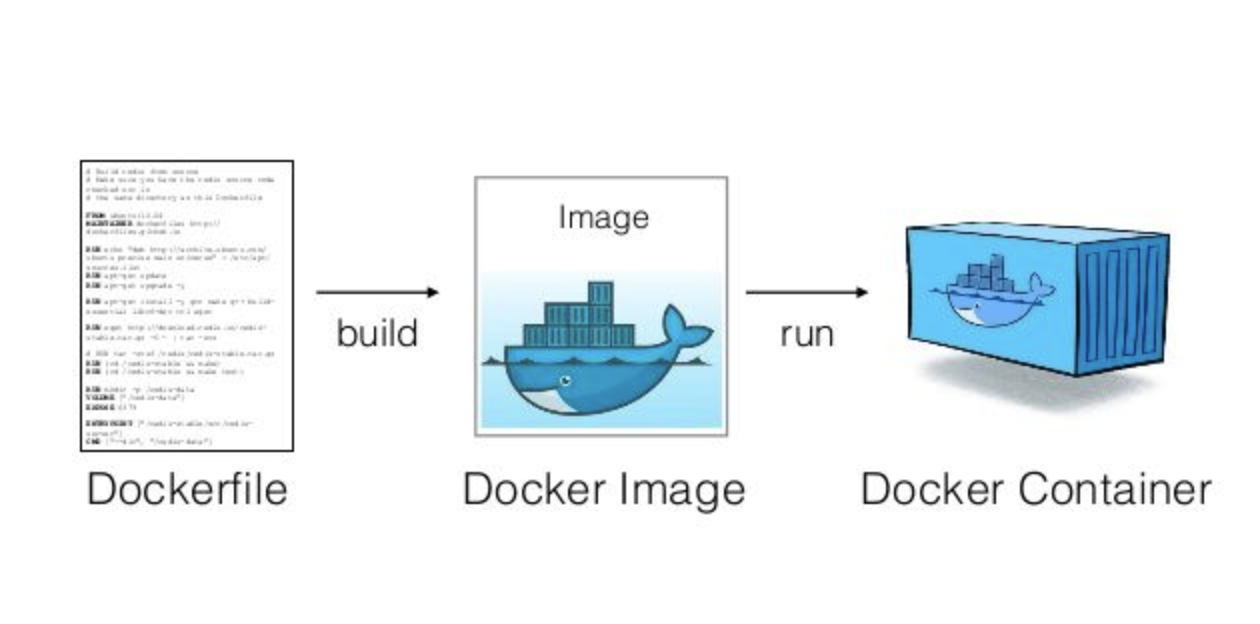
**Check the information regarding “Binds” & “Mounts”**

* + 1. **You can login to mysql container using this command**

**$ mysql -h 172.17.0.0 -u root -ppassword**

**$ show databases;**

**Build Docker Images**

****

**‘**

**Dockerfile Instructions**

* **FROM 🡪 Base Image**
* **LABEL 🡪 Adds metadata to an image**
* **RUN 🡪 Execute commands in a new layer and commit the results**
* **ADD/COPY 🡪 Adds files and folders into Image**
* **CMD 🡪 Runs binaries / commands on docker run**
* **ENTRYPOINT 🡪 Allows you to configure a container that will run as an executable**
* **VOLUME 🡪 Creates a mount point and marks it as holding externally mounted volumes**
* **EXPOSE 🡪 Container listens on the specified network ports at runtime**
* **ENV 🡪 Sets the environment variable**
* **USER 🡪 Sets the user’s name (or UID)**
* **WORKDIR 🡪 Sets the working directory**
* **ARG 🡪 Defines a variable that users can pass at build-time**
* **ONBUILD 🡪 Adds to the image a trigger instruction to be executed at a later time**

**Example 1:**

* **vim dockerfile**
  + **FROM jenkins/jenkins**
  + **MAINTAINER sscademy**
  + **USER root**
* **Create an image from the above dcokerfile**

**docker build -t myjenkins .**

* **Create a container from the above image**

**docker run --name j1 -d -P myjenkins**

* **Go into the interactive shell and check if the default user is root**

**docker exec -it j1 bash**

**whoami**

**Example 2:**

**1 Create a dockerfile**

**vim dockerfile**

**FROM ubuntu**

**MAINTIANER sscademy**

**ADD https://get.jenkins.io/war-stable/2.263.4/jenkins.war /**

**2 Create an image from the above dockerfile**

**docker build -t myubuntu .**

**4 Create a container from this image**

**docker run --name u1 -it myubuntu**

**5 Check if jenkins.war is present**

**Ls**

**Example 3:**

**1 vim dockerfile**

**FROM nginx**

**MAINTAINER sscademy**

**EXPOSE 90**

**2 Create an image**

**docker build -t mynginx .**

**3 Create a container from above image**

**docker run --name n1 -d -P mynginx**

**4 Check for the ports exposed**

**docker port n1**

**Example 3:**

**Create a dockerfile from ubuntu base image and make it behave**

**like nginx**

**1 Create a dockerfile**

**vim dockerfile**

**FROM ubuntu**

**MAINTAINER sscademy**

**RUN apt update**

**RUN apt install -y nginx**

**ENTRYPOINT ["/usr/sbin/nginx", "-g", "daemon off;"]**

**EXPOSE 80**

**2 Create an image from the above dockerfile**

**docker build -t myubuntu .**

**3 Create a container from the above image and it will work like nginx**

**docker run --name n1 -d -P myubuntu**

**4 Check the ports used by nginx**

**docker container ls**

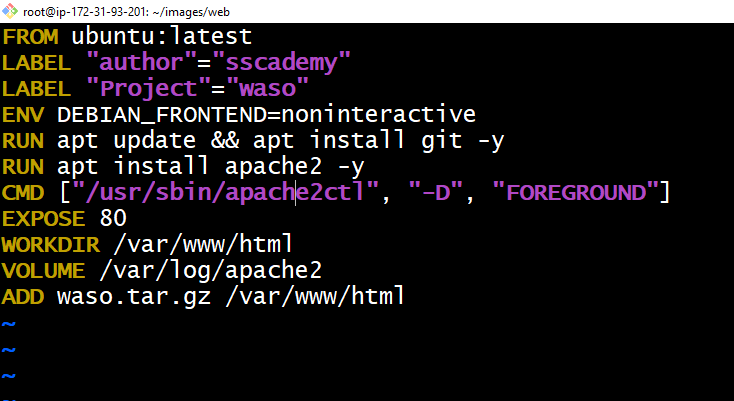
**5 To access nignx from browser**

**public\_ip\_of\_dockerhost:port\_no\_captured\_from\_step4**

**Let’s Deploy a Website in a Docker Container**

* **Login to Docker server**
* **Create a folder called Images 🡪 $ mkdir images**
* **Lets go inside the images folder 🡪 cd images**
* **Go to** [**www.tooplate.com**](http://www.tooplate.com)
* **Go to any website template 🡪 Clean Work**
* **Press F12**
* **Click on “Download button”**
* **Copy the Link**
* **Create a new directory 🡪 mkdir web**
* **Download using the link 🡪 wget** <https://www.tooplate.com/zip-templates/2132_clean_work.zip>
* **Now lets unzip it 🡪 install unzip software 🡪 sudo apt install unzip**
* **$ unzip 2132\_clean\_work.zip**
* **$ cd 2132\_clean\_work**
* **$ tar czvf cleanwork.tar.gz \***
* **$ mv cleanwork.tar.gz ../**
* **$ mv cleanwork.tar.gz /web**
* **Now lets write the Dockerfile**
* **Vim Dockerfile**
  + - **FROM ubuntu:latest**
    - **LABEL “Author”=”sscademy”**
    - **LABEL “Project”=”cleanwork”**
    - **ENV DEBIAN\_FRONTEND=noninteractive**
    - **RUN apt update**
    - **RUN apt install apache2 git -y**
    - **CMD [“/usr/sbin/apache2ctl”, “-D”, “FOREGROUND”]**
    - **EXPOSE 80**
    - **WORKDIR /var/www/html**
    - **VOLUME /var/log/apache2**
    - **ADD cleanwork.tar.gz /var/www/html**

tar –xvf documents.tar 🡪 for untar the tar.gz file



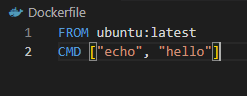
**Host the image into the Docker Hub**

* **Sign up Docker hub account**
* **Sign in to Docker hub account**
* **We have to build our image with our DockerHub account name**
  + **Docker build -t quadridevops135/image\_name .**
* **Now we can push the image to the DockerHub**
* **We need to login 🡪 $ docker login**
* **Give the DockerHub username 🡪 quadridevops135**
* **Password**
* **Now we can push the image 🡪 $ docker push quadridevops135/image\_name:tag**

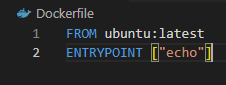
**ENTRYPOINT and CMD**

**We will be taking 3 Dockerfiles to understand the concept**

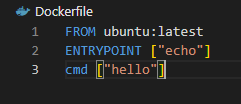
1. **CMD**



1. **ENTRYPOINT**



1. **ENTRYPOINT & CMD**



* 1. **Keep all the three Dockerfile in different folders and build the docker images with different tags**
  2. **Run the docker container and check the output by not supplying -d option**

**DOCKER COMPOSE**

**Docker compose is a tool, which is used to run multi-container architecture.**

* [**https://docs.docker.com/compose/**](https://docs.docker.com/compose/)

**Setup Docker compose**

[**https://docs.docker.com/compose/install/other/**](https://docs.docker.com/compose/install/other/)

1. **Download the docker compose package**

$ sudo curl -L "https://github.com/docker/compose/releases/download/1.29.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

1. **Give execute permissions on it**

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

1. **Verify it**

Docker-compose –version

**Example:**

[**https://docs.docker.com/compose/gettingstarted/**](https://docs.docker.com/compose/gettingstarted/)

**Step 1: Define the application dependencies**

1. Create a directory for the project:

$ mkdir composetest

$ cd composetest

**Step 2: Create a file called app.py in your project directory and paste the following code in**

import time

import redis

from flask import Flask

app = Flask(\_\_name\_\_)

cache = redis.Redis(host='redis', port=6379)

def get\_hit\_count():

retries = 5

while True:

try:

return cache.incr('hits')

except redis.exceptions.ConnectionError as exc:

if retries == 0:

raise exc

retries -= 1

time.sleep(0.5)

@app.route('/')

def hello():

count = get\_hit\_count()

return 'Hello World! I have been seen {} times.\n'.format(count)

**Step 3: Create another file called requirements.txt in your project directory and paste the following code in:**

flask

redis

**Step 4: Create a Dockerfile**

# syntax=docker/dockerfile:1

FROM python:3.7-alpine

WORKDIR /code

ENV FLASK\_APP=app.py

ENV FLASK\_RUN\_HOST=0.0.0.0

RUN apk add --no-cache gcc musl-dev linux-headers

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

EXPOSE 5000

COPY . .

CMD ["flask", "run"]

**Step 5: Define services in a Compose file**

version: "3.9"

services:

web:

build: .

ports:

- "8000:5000"

redis:

image: "redis:alpine"

**CONTAINERIZING THE PROJECT:**

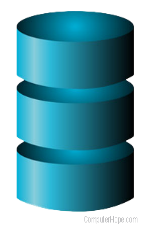
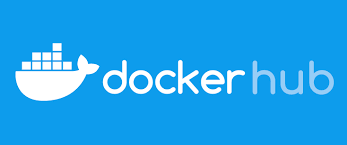
**Flow: -**

Dockerfile

Docker Build

Docker Compose

Docker Push



Docker Images

