```
# git clone https://github.com/sinhakiara/edbda123.git

# cd edbda123/

# cat > username.txt
NAME

# git config --global user.name "NAME"
# git config --global user.email "EMAIL"

# git add .
# git commit -m "MSG"

# git branch

# git status

# git log

# git push -u origin main

username: sinhakiara
password: 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
```

install on AWS/Ubuntu:

` ` ` ` `

```
# apt update
# apt install docker.io -y
```

Check:

` `

```
# systemctl status docker
# docker --help
```

Image :

`

docker hub [public/private repo server]

Docker Commands:

----

1) run : Start a container

` ` ` ` `

```
# docker run image
OR
# docker run image:latest
# docker run image:1.1.0
# docker run -d image:1.1.0
```

[-d: detach]

2) ps : list of running containers

```
# docker ps
# docker ps -a [ running & stopped containers]
# docker ps -q [ Quite : list the ID of container ]
# docker ps -aq
```

3) stop : Stop a container

```
# docker stop NAME
                  or
              CONTAINER ID
```

```
# docker stop $(docker ps -q)
```

```
# docker start CONTAINER_ID
# docker stop CONTAINER_ID
# docker restart CONTAINER_ID
```

----

4) rm : Remove a container

```
# docker rm name/container ID [ Remove a stopped container ]
# docker rm -f name/container ID
```

eg:

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

5) images : list images

```
# docker images
# docker images -q
```

PULL:

~~~

```
# docker pull image
# docker pull image:tag
```

6) rmi : Remove image [-f : forcefully]

```
# docker rmi <image_id>
# docker rmi -f <image_id>
```
~~~

eg:

```
# docker rmi $(docker images -q)
# docker rmi -f $(docker images -q)
```

7) pull : Only download image

```
# docker pull docker/whalesays
```

8) exec - execute command

```
# docker exec <name> cat /etc/passwd
# docker exec -it <name/id> bash
```

9) run : attach & detach

```
# docker run image [Attach with terminal]
```

```
# docker run -d image [ Detach ]
# docker attach c6ecf [Attach again]
```

Q. How many containers are running on this host?

```
# docker ps
```

Q. How many images are available on this host?

```
# docker images
```

Q. Run a container using the redis image

Image:redis

```
# docker run -d redis
```

Q. Stop the container named as redis.

```
# docker ps
# docker stop image_id
```

Q. How many containers are PRESENT on the host now? Including both Running and Not Running ones !

```
# docker ps -a
```

Q. Delete all containers from the Docker Host.

Both Running and Not Running ones. Remember you may have to stop containers before deleting them.

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

Q. Cleanup: Delete all images on the host

Remove containers as necessary

```
# docker rmi -f $(docker images -q)
```

___

Q. You are required to pull a docker image which will be used to run a container later. Pull the image nginx:1.14-alpine

Only pull the image, do not create a container.

```
# docker pull nginx:1.14-alpine
```

Q. Run a container with the nginx:1.14-alpine image and name it webapp

```
# docker run --name "webapp" -d nginx:1.14-alpine
```

Display the docker host information with:

~~~

```
# docker info
```
~~~

~~~
# docker run -it ubuntu:latest bash
~~~

-i [input]
-t [Prompt terminal]

~~~
# docker exec -it a0aaac555c0e bash
~~~

INSPECT:

~~~
```
~~~

IP:

~~~
`
~~~

~~~
# docker inspect -f '{{json .NetworkSettings.IPAddress}}' container_ID/Name
~~~

GW:

~~~
# # docker inspect -f '{{json .NetworkSettings.Gateway}}' container_ID/Name
~~~

Port mapping:

----

* Lets run a single container with a simple web application (httpd:80) on port 8080 on host machine.

```
# docker run -d --name "web1" -p 8080:80 httpd
```

Image: nginx

Port: 80/tcp

Map: 8080<->80/tcp

Name: webserver1

```
curl "http://VM'sIP:8080
```

```
# docker run -d --name "webserver1" -p 8080:80 nginx:latest
```

# systemctl: This command is to handle Linux services.

start, stop, restart, reload, force-reload, status

Volume Mapping:

```
container httpd: /usr/local/apache2/htdocs
Host OS : /web1

# mkdir /web1
# cat > /web1/index.html
<html><body><h1>ULALA</h1></body></html>
```

```
# docker run -d --name "server1" -v
/web1:/usr/local/apache2/htdocs httpd:latest
```

```
# mkdir /web1 /web2 /web3
```

Create index page

```
# docker run -d -p 80:80 --name web1 -v /web1:/usr/local/apache2/htdocs httpd
```

```
# docker run -d -p 81:80 --name web2 -v /web2:/usr/local/apache2/htdocs httpd
# docker run -d -p 82:80 --name web3 -v /web3:/usr/local/apache2/htdocs httpd
```

```
# docker run -d -p 1000-2000:80 --name web3 -v
/web3:/usr/local/apache2/htdocs httpd
```

```
# docker run -d -p 8080:80 -v /opt/data:/usr/local/apache2/htdocs httpd
```

Inspect Container:

```
# docker inspect container_name
```

It return the data in JSON format

____-

Q. Create your own image which can run a basic Node.js web server as following:

- Use Image: mhart/alpine-node:4.4

- Use your favourite text editor to add app.js:

````Code Snippet Start````

```
var http = require('http');
http.createServer(function (req, res) {
  console.log(new Date().toUTCString() + " - " + req.url);

  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello, Docker.\n');
}).listen(3000);

console.log('Server running at http://0.0.0.0:3000/');
```

````Code Snippet End````

- Create an entrypoint with the command:

```
 /usr/bin/node app.js
```

Q. Deploy an app using python Flask server & create Dockerfile to build image as follwing:

- Install all required dependencies

- Install Flask

```
pip install flask
```

- The code "app.py":

```
import os
from flask import Flask
app = Flask(__name__)

@app.route("/")
def main():
    return "Welcome!"

@app.route('/hackers')
def hello():
    return 'Hey buddy, how are you?'

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080)
```

- Entrypoint to Start Web Server:

python3 app.py

Docker in DevOps Engineer:

```
-------------                                              ------------
| Developer |  --->  app.py & Guide/README/manual  ------>  | Ops Team | :(
-------------                                              ------------
-------------                                                   ------------
| Developer |  ---> app.py & Guide -----> Docker Image  ------>  | Ops Team | :)
-------------                                                   ------------
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Dockerfile:

Image: ubuntu:latest

```
apt update
apt install python3 -y
python3 -m http.server 5000
```

```
# vim Dockerfile

FROM ubuntu:latest

RUN apt update
RUN apt install python3 -y

COPY index.html /opt/index.html

EXPOSE 5000

ENTRYPOINT python3 -m http.server 5000
```

OR

NOTE: CMD defines the commands that will run on the Image at start-up

CMD ["python3", "./app.py"]

CMD ["/bin/bash", "echo", "Hello World"]

# docker build . -t dheeraj/webapp007

# docker save id > file.tar

# docker import file.tar
# docker images

Docker Push:
```
```

```
# docker login
username: adm
password: ***
```

```
# docker push dheeraj/webapp007
```

You can search it : hub.docker.com

```
# cat > Dockerfile
FROM busybox
MAINTAINER Dheeraj (email@domain.com)
ENTRYPOINT ["/bin/cat"]
CMD ["/etc/passwd"]

# docker build . -t dheeraj/fun
# docker push dheeraj/fun
# docker run -it myimage
```

Designing a Dockerfile & docker-compose for a web-server (hello_node)

\`\`\`\`\`\`\`

- create your own image which can run a basic Node.js web server

- Create a new directory and then use your favourite text editor to add app.js

```
  Image: mhart/alpine-node:4.9
  # mkdir hello_node
  # cd hello_node
  # /usr/bin/node app.js
```

- app.js

```
var http = require('http');
http.createServer(function (req, res) {
  console.log(new Date().toUTCString() + " - " + req.url);

  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello, Docker.\n');
}).listen(3000);

console.log('Server running at http://0.0.0.0:3000/');
```

- Now design the Dockerfile accordingly.

EG:

Simple Web Application:

`````

simple web application using Python Flask and MySQL database.

Below are the steps required to get this working on a base linux system:

- Install all required dependencies
- Install and Configure Web Server
- Start Web Server

1. Install all required dependencies

```````

~~~
# apt install -y python3 python3-setuptools python3-dev build-essential
python3-pip python3-mysqldb
~~~

2. Install and Configure Web Server

``````

~~~
# pip3 install flask
# pip3 install flask-mysql
~~~

3. Create web app:

```

~~~
# nano app.py
~~~

```
import os
from flask import Flask
app = Flask(_name_)

@app.route("/")
def main():
    return "Welcome!"

@app.route('/ulala')
def hello():
    return 'Hey babe, how are you?'

if _name_ == "_main_":
    app.run(host="0.0.0.0", port=8080)
```

4. Start Web Server:

~~~

```
# FLASK_APP=app.py flask run --host=0.0.0.0
```

5. Test:

``

```
http://<IP>:5000                 => Welcome
http://<IP>:5000/ulala           => Hey babe, how are you?
```

=========
~~~

TASKS:

```
apt update
apt install python3 python3-pip
pip3 install flask
Create/Copy application code to /opt/app.py

FLASK_APP=/opt/app.py flask run --host=0.0.0.0
```

SOLUTION:

```
# mkdir ./myapp
# cd ./myapp
# cat > Dockerfile
FROM ubuntu

RUN apt update
RUN apt install python3 python3-pip
RUN pip3 install flask

COPY app.py /opt/app.py

ENTRYPOINT FLASK_APP=/opt/app.py flask run --host=0.0.0.0
```

```
# docker build . -t mysampleapp
# docker images
# docker run mysampleapp
```

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

YAML Theory & Lab

``

YAML:

-----

1. [key-value] pair:

```
key: value
```

eg:

```
Fruit: Apple
veg: carrot
```

2. Array/list:

```
Fruit:
        - orange
        - apple
```

3. Dict
``
Its a set properties taht group together:

```
Banana:
        calories: 102
Grapes:
        calories: 99
```

[+] key value/dict/list:

```
----
Fruits:
        - Banana:
                calories: 102
        - grapes:
                calories: 99
```

-----------------------------------------------------
Docker Compose [yaml/yml]
--------------------------

eg:

```
# docker run httpd:2
# docker run nginx
# docker run ansible
# docker run httpd:latest
```

OR

```
# vim docker-compose.yml

services:
        proxy:
                image: "nginx"
        orch:
                image: "ansible"
        web2:
                image:        "httpd:latest"
```

With port:

~~~

```
# vim docker-compose.yml

version: "3.0"
services:
        web1:
```
~~~

```
                image: "httpd"
                ports:
                        - "80:80"

                environment:
                        - var=value
        web2:
                image: "nginx"
                ports:
                        - "1000-2000:81"
        web3:
                image: python:3.9
                ports:
     - 8000:8000
                working_dir: /opt
                container_name: web3
                command: python3 -m http.server 8000

# docker-compose up
or
# docker-compose up -d

# docker-compose ps
----------------------------------------
# apt install docker-compose -y
# vim docker-compose.yaml
services:
```

```
webapp1:
        image: httpd:latest
        ports:
                - 1001:80
        volumes:
                - /webapp1:/usr/local/apache2/htdocs

webapp2:
        image: httpd:latest
        ports:
                - 1002:80
        volumes:
                - /webapp2:/usr/local/apache2/htdocs

webapp3:
        image: httpd:latest
        ports:
                - 1003:80
        volumes:
                - webapp3:/usr/local/apache2/htdocs

webapp4:
        image: nginx:latest
        ports:
                - 1004:80
        volumes:
                - webapp4:/usr/share/nginx/html
```

```
  webapp5:
    image: nginx:latest
    ports:
      - 1005:80
    volumes:
      - webapp5:/usr/share/nginx/html
```

---

https://prezi.com/_vhtsx-u9qgq/serverless-architecture-aws-lambda/

Another Example of Lambda Serverless Functions:
~~~

s3 bucket trigger:
`````````````````

1. Goto service IAM -> Roles -> Create Role -> In Trusted entity type [select : AWS service] -> In Use case [select: Lambda] -> Next -> In Permissions policies [search for "S3" and select "AmazonS3FullAccess"] AND [search for "cloudwatchfullaccess" and select "CloudWatchFullAccessV2"] AND [search for "LambdaBasic" and select "AWSLambdaBasicExecutionRole"]-> Next -> Role name ["s3s3s3s3"] -> Create Role
~~~

2. Create Lambda function from scratch with python env -> In "Change default execution role" -> Select "Use an existing role" named as "s3s3s3s3" (for this example) -> Create Function

Code snippet start

```


def lambda_handler(event, context):
    print("Lambda Triggereddddddddddddddddddd")
    return {
        print('Hello from Lambda!')
    }

```

Code snippet end

3. Create Access & Secret Key to access AWS via AWSCLI

4. Create S3 bucket

5. Go to Lambda function & create Trigger -> Trigger configuration [search for "S3"] -> select the S3 bucket name "bucket_fun_lambda" -> Event types [select "All object create events" for this example] -> Add

6. # aws s3 cp localfile.txt s3://bucket_fun_lambda

7. To verify: Go to "Cloudwatch" service -> Log groups -> Select your log group -> /aws/lambda/<lambda_fun_name> -> check logs

Another Example of Lambda Serverless Functions:

````````

s3 bucket trigger:

```

1. Goto service IAM -> Roles -> Create Role -> In Trusted entity type [ select : AWS service ] -> In Use case [select: Lambda] -> Next -> In Permissions policies [search for "S3" and select "AmazonS3FullAccess" ] AND [search for "cloudwatchfullaccess" and select "CloudWatchFullAccessV2" ] AND [search for "LambdaBasic" and select "AWSLambdaBasicExecutionRole" ]-> Next -> Role name [ "s3s3s3s3" ] -> Create Role

2. Create Lambda function from scratch with python env -> In "Change default execution role" -> Select "Use an existing role" named as "s3s3s3s3" (for this example) -> Create Function

Code snippet start

~~~
def lambda_handler(event, context):
    print("Lambda Triggereddddddddddddddddddd")
    return {
~~~

```
        print('Hello from Lambda!')
    }
```

Code snippet end

3. Create Access & Secret Key to access AWS via AWSCLI

4. Create S3 bucket

5. Go to Lambda function & create Trigger -> Trigger configuration [search for "S3"] -> select the S3 bucket name "bucket_fun_lambda" -> Event types [select "All object create events" for this example] -> Add

6. # aws s3 cp localfile.txt s3://bucket_fun_lambda

7. To verify: Go to "Cloudwatch" service -> Log groups -> Select your log group -> /aws/lambda/<lambda_fun_name> -> check logs