DEV OPS

Practical-2

Create Dockerfile:
$ cat Dockerfile

```
# Use the official Ubuntu 18.04 as base
FROM ubuntu:18.04
# Install nginx and curl
RUN apt-get update &&
apt-get upgrade -y &&
apt-get install -y nginx curl &&
rm -rf /var/lib/apt/lists/*
```

Question-1:

Create docker image from above Dockerfile and push into dockerhub repository. Image should tag :V3
Note: use docker build command with below flags:
 -- rm=true; all intermediate containers are removed after a successful build

-- no-cache=false; avoids using cache during build

Question-2
Mini App:

First, create a new directory where all the files would live. In this directory create a package.json file that describes your app and its dependencies:

```
{
  "name": "docker_web_app",
  "version": "1.0.0",
  "description": "Node.js on Docker",
  "author": "First Last <first.last@example.com>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.16.1"
  }
}
```

With your new package.json file, run npm install. If you are using npm version 5 or later, this will generate a package-lock.json file which will be copied to your Docker image.

Then, create a server.js file that defines a web app using the Express.js framework:

$ npm install express –save

```
'use strict';


const express = require('express');


// Constants
const PORT = 8080;
const HOST = '0.0.0.0';


// App
const app = express();
app.get('/', (req, res) => {
res.send('Hello World');
});


app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

Finally, Dockerfile should look like this

```
FROM node:14


# Create app directory
WORKDIR /usr/src/app


# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./


RUN npm install
# If you are building your code for production
# RUN npm ci --only=production
# Bundle app source
COPY . .


EXPOSE 8080
CMD [ "node", "server.js" ]
```

Create a .dockerignore file in the same directory as your Dockerfile with following content:

```
node_modules
npm-debug.log
```

This will prevent your local modules and debug logs from being copied onto your Docker image and possibly overwriting modules installed within your image.

Building your image

Go to the directory that has your Dockerfile and run the following command to build the Docker image. The -t flag lets you tag your image so it's easier to find later using the docker images command:

```
docker build . -t <your username>/node-web-app
```

Your image will now be listed by Docker:

```
$ docker images


# Example
REPOSITORY                     TAG        ID              CREATED
node                       14         1934b0b038d1 5 days ago
<your username>/node-web-app    latest     d64d3505b0d2 1 minute ago
```

Run the image

Running your image with -d runs the container in detached mode, leaving the container running in the background. The -p flag redirects a public port to a private port inside the container. Run the image you previously built:

```
docker run -p 49160:8080 -d <your username>/node-web-app
```

Print the output of your app:

```
# Get container ID
$ docker ps


# Print app output
$ docker logs <container id>
```

```
# Example
Running on http://localhost:8080
```

If you need to go inside the container you can use the exec command:

```
# Enter the container
$ docker exec -it <container id> /bin/bash
```

Test

To test your app, get the port of your app that Docker mapped:

```
$ docker ps


# Example
ID           IMAGE                              COMMAND     ... PORTS
ecce33b30ebf <your username>/node-web-app:latest npm start ... 49160->8080
```

In the example above, Docker mapped the 8080 port inside of the container to the port 49160 on your machine.

Now you can call your app using curl (install if needed via: sudo apt-get install curl):

```
$ curl -i localhost:49160


HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 12
ETag: W/"c-M6tWOb/Y57lesdjQuHeB1P/qTV0"
Date: Mon, 13 Nov 2017 20:53:59 GMT
Connection: keep-alive


Hello world
```

Question 3:-

Publish Port in the Docker Container.

Command:-

```
docker container run --rm --name nginx \
        --publish
target=80,published=127.0.0.1:8081,protocol=tcp \
```

-d nginx

Another Way:
docker container run --rm --name nginx \
          -p 80:127.0.0.1:8081/tcp -d nginx

Check Publish port:
docker container port <container-name>

Question 4:-

a) Docker run to go inside the busybox image container

b)Create directory and file

Dir: mkdir -p /data/html

File: vi /data/html/index.html

c) Add some contents in the 'index.html' file

d) Commit the container and create image 'busyboxweb'

Use below docker commit:

docker commit -p -a "champu <champu@gmail.com>" -m "increase index.html Webpage" <container-name>   <newimage-name>

# -a --author Is the author of the specified change
# -c --change list application dockerfile modify CMD command
# -m --message Record the content of this modification
# -p Pause container during commit (default true)

e)Docker inspect busybox

Look : CMD section

f) Run container from new-image which created in (e) step

Run inside the container:

# httpd -h

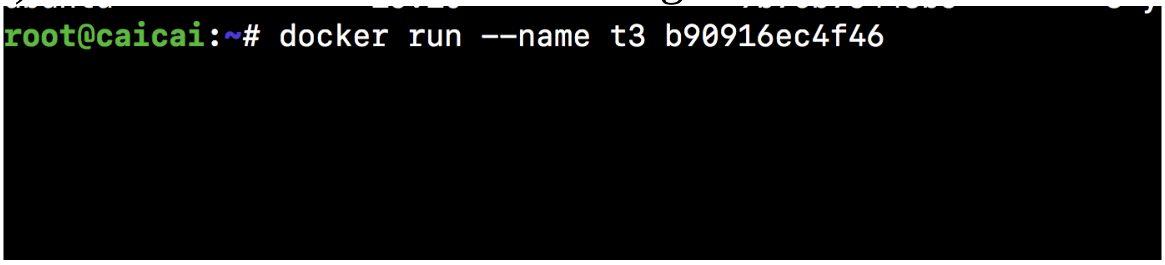
g) Modify CMD command:

docker commit -a "champu<champu@gmail.com>" -c 'CMD

["/bin/httpd","-f","-h","/data/html"]' -p t2 champu/httpd:V0-1-1-3

# -a Designated author
# -c modify cmd Command in the form of cmd list
# -p Execution container

h) Run the container from new image

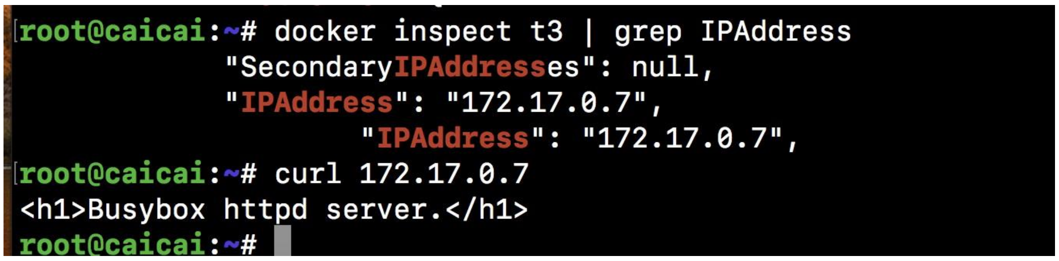


```
root@caicai:~# docker run --name t3 b90916ec4f46
```

i) Access the container

docker inspect t3

# curl <container-ip>



```
root@caicai:~# docker inspect t3 | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "172.17.0.7",
                    "IPAddress": "172.17.0.7",
root@caicai:~# curl 172.17.0.7
<h1>Busybox httpd server.</h1>
root@caicai:~#
```

Question 5:-

# vi index.html
<h2>It Works!</h2>
<h3>Welcome to Alnafi</h3>

Dockerfile

```
FROM tecadmin/ubuntu-ssh:16.04
LABEL maintainer="champu@gmail.com"

RUN apt-get update \
  && apt-get install -y apache2

COPY html/* /var/www/html/

WORKDIR /var/www/html
```

```
CMD ["apachectl", "-D", "FOREGROUND"]
EXPOSE 80
```

Question 6:

Try below command with and without -d flag, use docker inspect, docker logs command and make the lighter as well.

```
docker run ubuntu:18.04 /bin/sh -c "while true; do echo hello world; sleep 1; done"
```

Question 7:

clean up all terminated containers with the following command.

```
$ docker container prune
```

Use the following command to batch delete all the exited containers

```
$ docker rm -v $(docker ps -aq -f status=exited)
```

Question 8:-

a) Pull image busybox:latest
b) Run the container and create file file.txt
c) Create image from running container and tag with version1
d) Run the container from new Image and verify the changes
e) Create new image from step-d container, it should be version2

Use this command

```
# docker commit --change “ENV DEBUG=true”
```

<container-name> <Imagename:version2>

f) Run this command:

```
# docker inspect -f "{{ .Config.Env }}"
<Imagename:version2>
```

Question 9:-

a) Take any base image and create basic Dockerfile and create image
b) Need to create Three images with 3 different tags
c) Push all docker images in the docker hub
d) Delete all images from your local machine
e) Now pull images from dockerhub with specific version
f) Again delete same image which you pulled.
g) Now while pulling docker images try below commands

```
# docker pull –all-tags <imagename>
```

Question10:-

Your manager asked to deploy Jenkins server using Docker Technology for application build and deployment.
He asked you to use vulnerable free and official Jenkins image and he wants to security purpose run Jenkins non-default port.
Once Jenkins server up and running, you need to share Jenkins's server IP address with your development team so that they can start application build.