Day 17: Docker Project

This is #90DaysofDevops challenge under the guidance of Shubham Londhe sir.

Day 17 TASK

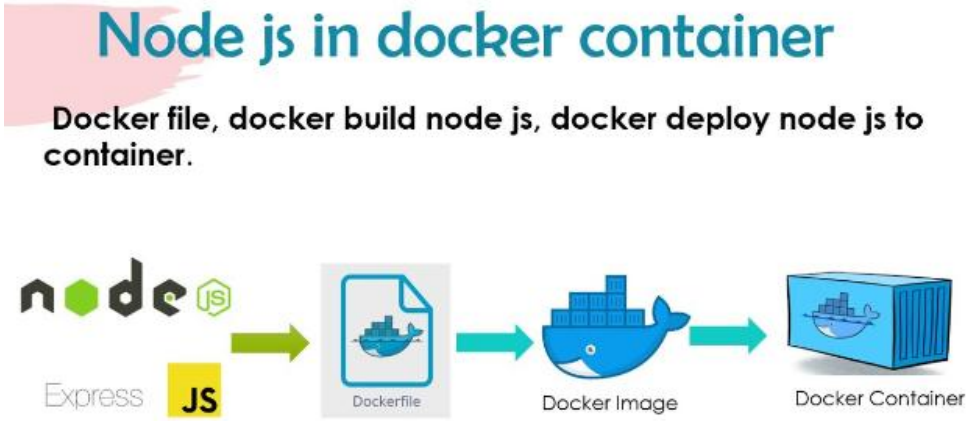
check this for task:

https://github.com/LondheShubham153/90DaysOfDevOps/blob/master/2023/day17/tasks.md

Dockerfile

Docker is a tool that makes it easy to run applications in containers. Containers are like small packages that hold everything an application needs to run. To create these containers, developers use something called a Dockerfile.

A Dockerfile is like a set of instructions for making a container. It tells Docker what base image to use, what commands to run, and what files to include. For example, if you were making a container for a website, the Dockerfile might tell Docker to use an official web server image, copy the files for your website into the container, and start the web server when the container starts.



TASKS:

- **Create a Dockerfile for a simple web application (e.g. a Node.js or Python app)**

Before moving forward with the creation of Dockerfile, you need to install docker in your server. We have seen the steps to install docker in the previous article (Day 16).

Link is below:

https://swapnasagarpradhan.medium.com/how-to-install-docker-on-amazon-linux-2-8e5161ac5464

- **Build the image using the Dockerfile and run the container**
- **Verify that the application is working as expected by accessing it in a web browser**
- **Push the image to a public or private repository (e.g. Docker Hub)**

For this you first need a AWS EC2 instance of your own choice.

Here is my blog on creating AWS EC2 instance :

https://medium.com/@misalPav/launching-your-first-ec2-instance-a422862e09c2

Once you are done with creating a blog you need to SSH into it.

Below are the steps that we will be going to perform in the process:

1. Install Git and clone the repo of the Node.js application
2. Install Docker
3. Create and configure a Dockerfile
4. Build a Docker image
5. Create and run a Docker container
6. Access it

1. Clone the repo of the Node.js application

Open the instance, first you need to install Git in it so that we can clone the application repository from the GitHub(VCS). Use command :

```
#yum install git -y
```

```
root@ip-172-31-57-178:/home/ec2-user
[root@ip-172-31-57-178 ec2-user]# yum install git -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
---> Package git.x86_64 0:2.38.1-1.amzn2.0.1 will be installed
--> Processing Dependency: perl-Git = 2.38.1-1.amzn2.0.1 for package: git-2.38.1
-1.amzn2.0.1.x86_64
--> Processing Dependency: git-core-doc = 2.38.1-1.amzn2.0.1 for package: git-2.
38.1-1.amzn2.0.1.x86_64
--> Processing Dependency: git-core = 2.38.1-1.amzn2.0.1 for package: git-2.38.1
-1.amzn2.0.1.x86_64
--> Processing Dependency: perl(Term::ReadKey) for package: git-2.38.1-1.amzn2.0
.1.x86_64
--> Processing Dependency: perl(Git::I18N) for package: git-2.38.1-1.amzn2.0.1.x
86_64
--> Processing Dependency: perl(Git) for package: git-2.38.1-1.amzn2.0.1.x86_64
```

Now clone the application repository, using the command :

```
#git clone https://github.com/rajani103/nodejs-on-ec2.git
(git URL of repo)
```

```
root@ip-172-31-57-178:/home/ec2-user
[root@ip-172-31-57-178 ec2-user]# git -v
git version 2.38.1
[root@ip-172-31-57-178 ec2-user]# git clone https://github.com/rajani103/nodejs-
on-ec2.git
Cloning into 'nodejs-on-ec2'...
remote: Enumerating objects: 77, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 77 (delta 3), reused 0 (delta 0), pack-reused 68
Receiving objects: 100% (77/77), 13.43 KiB | 2.69 MiB/s, done.
Resolving deltas: 100% (30/30), done.
[root@ip-172-31-57-178 ec2-user]#
```

2. Install Docker

Install Docker in the machine using the command :

```
#yum install docker -y
```

```
root@ip-172-31-57-178:/home/ec2-user
[root@ip-172-31-57-178 ec2-user]# yum install docker
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core                                               | 3.7 kB     00:00
Resolving Dependencies
--> Running transaction check
---> Package docker.x86_64 0:20.10.17-1.amzn2.0.1 will be installed
--> Processing Dependency: runc >= 1.0.0 for package: docker-20.10.17-1.amzn2.0.
1.x86_64
--> Processing Dependency: libcgroup >= 0.40.rc1-5.15 for package: docker-20.10.
17-1.amzn2.0.1.x86_64
--> Processing Dependency: containerd >= 1.3.2 for package: docker-20.10.17-1.am
zn2.0.1.x86_64
--> Processing Dependency: pigz for package: docker-20.10.17-1.amzn2.0.1.x86_64
--> Running transaction check
---> Package containerd.x86_64 0:1.6.8-1.amzn2 will be installed
```

Now check the version of the docker once and start the docker and check the status of the docker to know if it is running using the below commands :

```
#docker -v
#systemctl start docker
#systemctl status docker
```

```
root@ip-172-31-57-178:/home/ec2-user
[root@ip-172-31-57-178 ec2-user]# docker -v
Docker version 20.10.17, build 100c701
[root@ip-172-31-57-178 ec2-user]# systemctl start docker
[root@ip-172-31-57-178 ec2-user]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor pres
et: disabled)
   Active: active (running) since Wed 2023-01-18 14:37:22 UTC; 5s ago
     Docs: https://docs.docker.com
  Process: 3545 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=
exited, status=0/SUCCESS)
  Process: 3544 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SU
CCESS)
 Main PID: 3548 (dockerd)
    Tasks: 7
   Memory: 21.6M
   CGroup: /system.slice/docker.service
           └─3548 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/cont...
```

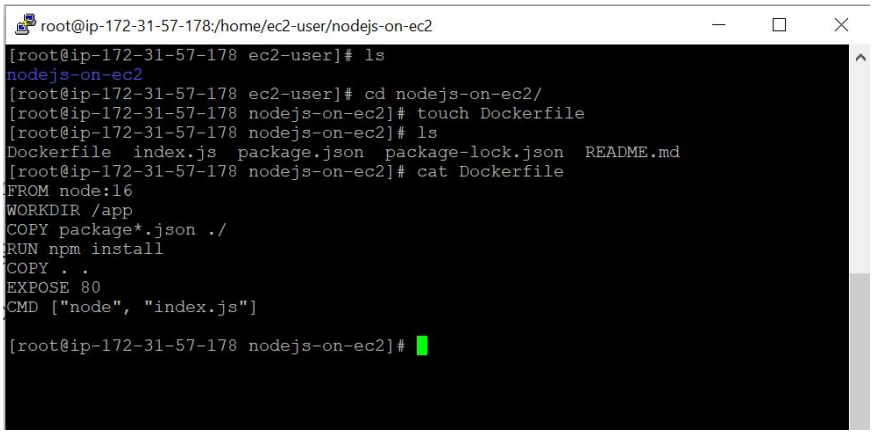
3. Create and configure a Dockerfile

Now we will create and configure a dockerfile as per the requirement of the Node.js application. Change the directory to the cloned project and create Dockerfile there.

Here is the Dockerfile that I have created :

```
FROM node:16
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 80
CMD ["node", "index.js"]
```

You can check my previous blog for understanding the Dockerfile.



4. Build a Docker image

Now before starting the build process check if there is any existing container running with the same name.

Use command :

```
#docker ps
#docker ps -a
```

Now you are all set to build the image. Use command :

```
# docker build . -t app
```

```
[root@ip-172-31-57-178 nodejs-on-ec2]# docker build . -t app
Sending build context to Docker daemon  83.46kB
Step 1/7 : FROM node:16
16: Pulling from library/node
ac7f2e1c7586: Pull complete
dbcdf7fce05b: Pull complete
0ed0c2752d84: Pull complete
bf01cd4ea334: Pull complete
739282cf09da: Extracting  118.7MB/191.9MB
64c938dc9431: Download complete
3cdccd57d93f: Download complete
1e0d12324fde: Download complete
0537aadad17e: Download complete
```

```
 ---> 887bd9dfa934
Step 6/7 : EXPOSE 80
 ---> Running in 3509259b3dec
Removing intermediate container 3509259b3dec
 ---> 47bcf8db87a7
Step 7/7 : CMD ["node", "index.js"]
 ---> Running in e269ef8a3025
Removing intermediate container e269ef8a3025
 ---> f71fd51184bd
Successfully built f71fd51184bd
Successfully tagged app:latest
[root@ip-172-31-57-178 nodejs-on-ec2]#
```

5. Create and run a Docker container

Using the image that has been built we will create a container out of it and run it:

Use the below commands :

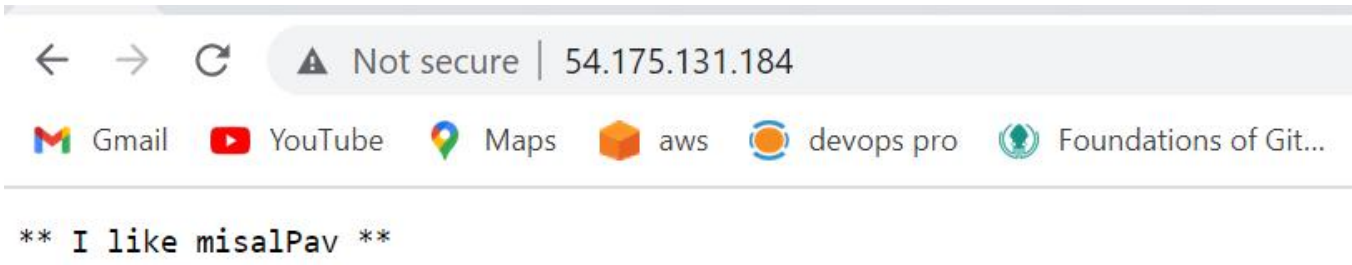
```
# docker run -d --name nodejs-app-cont -p 80:80 app:latest
```

You can see a container running here which can be accessed on port 80 as we have done the port mapping on port 80.

```
[root@ip-172-31-57-178 nodejs-on-ec2]# docker run -d --name nodejs-app-cont -p
0:80 app:latest
e59e1285acf24525d3378cc115320f6db4d3d950316abc5b2ba16d780ec5e4c8
[root@ip-172-31-57-178 nodejs-on-ec2]# docker ps
CONTAINER ID   IMAGE        COMMAND                  CREATED         STATUS
    PORTS                                   NAMES
e59e1285acf2   app:latest   "docker-entrypoint.s…"   7 seconds ago   Up 6 secon
s   0.0.0.0:80->80/tcp, :::80->80/tcp   nodejs-app-cont
```

6. Access it

Now you can take the public IP of the machine and port 80 to access the application.



And yess!! it is accessible.

7. Pushing the image on DockerHub

We have already created a Docker image using the Dockerfile. Check all the images present by using the command.

```
docker images
```

Now we will login into the DockerHub by adding the command:

```
Docker login
```

Put your username and password here. If the login is successful, then you will get a message like **Login Succeeded**.

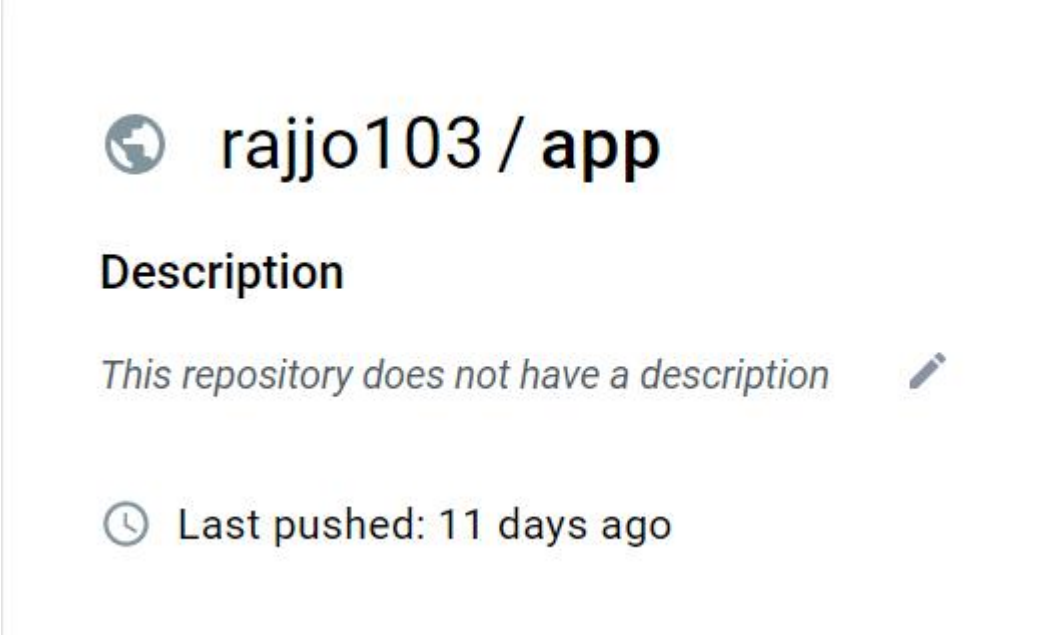
Now tag the locally created image to the docker hub. This means we have to tag the image with the docker hub username.

```
docker tag app:latest rajjo103/app:latest
```

Now push the image to the Docker hub using the push command.

```
docker push rajjo103/app:latest
```

And it is done, you can check in your Docker hub if it is pushed.



Please, feel free to drop any questions in the comments below. I would be happy to answer them.

If this post was helpful, please do follow and click the clap

_Thank you for reading

_Rajani