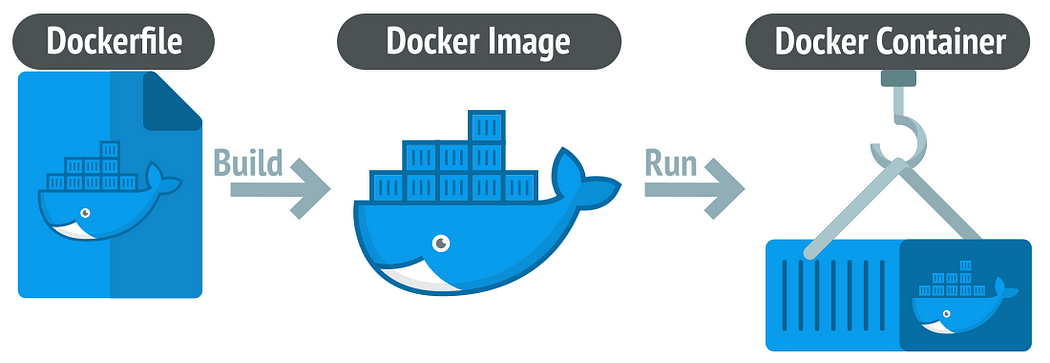
Hello there,

We will containerize a node.js application using Docker and access the same on the server.



Before starting with the process have an introduction on:

* Docker
* Docker Container
* Docker Image
* Dockerfile
* DockerHub

Now we will start the containerization process of the Node.js application.

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

Once you are done with creating an instance you need to SSH into it.

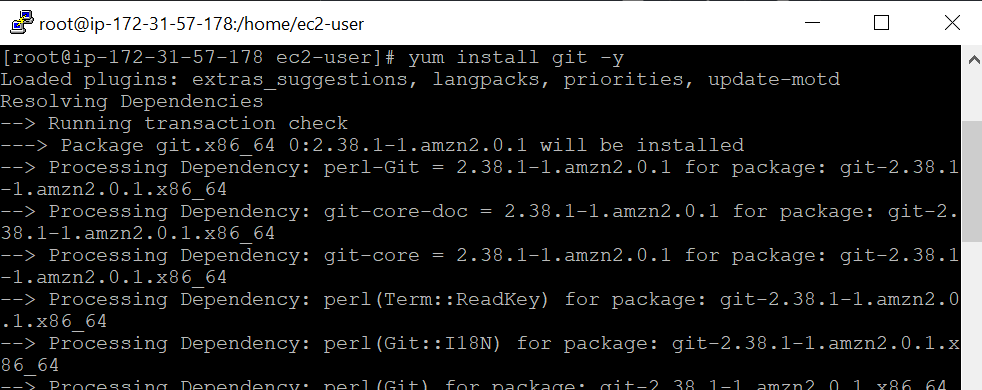
Below are the steps that we will be going to perform in the process:

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**

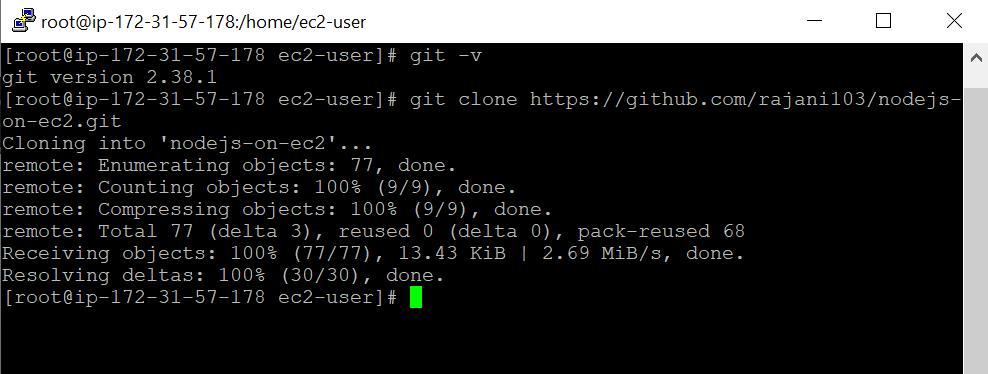
Open the instance, first you need to install Git in it so that we can clone the application repository from the GitHub(VCS). Use command :

#yum install git -y



Now clone the application repository, using the command :

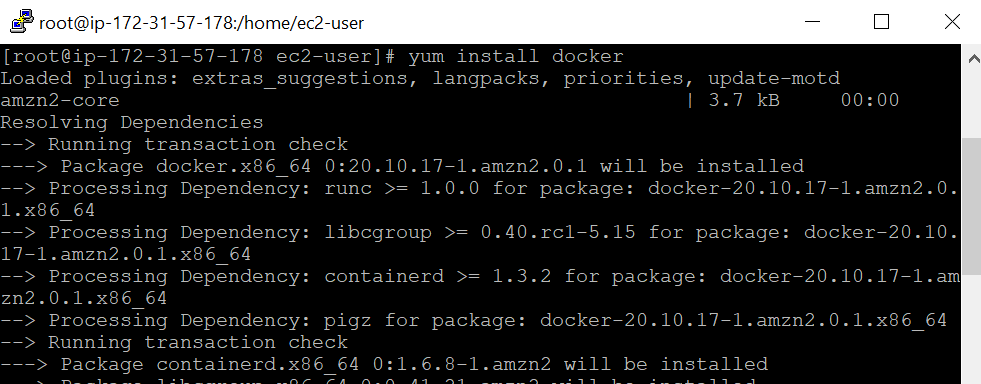
#git clone https://github.com/rajani103/nodejs-on-ec2.git (git URL of repo)



## **2. Install Docker**

Install Docker in the machine using the command :

#yum install docker -y

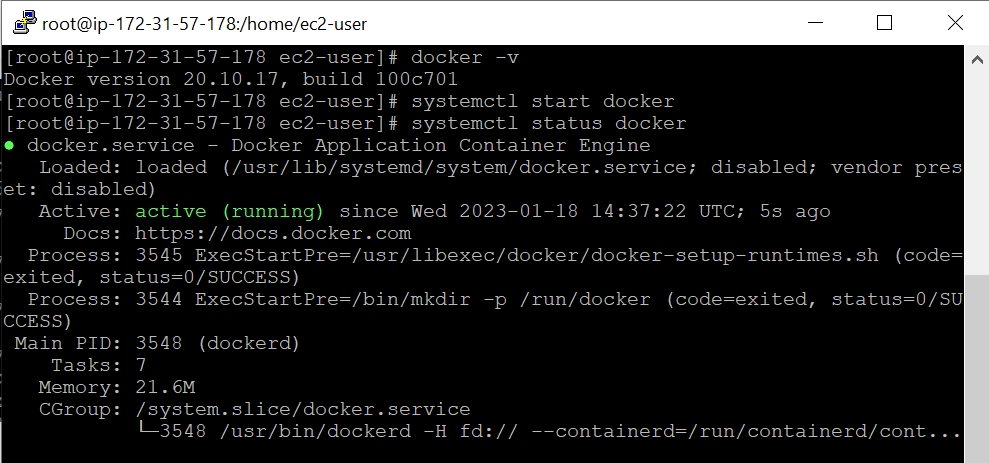


Now check the version of the docker once and start the docker and check the status of the docker to know if it is running using the below commands :

#docker -v

#systemctl start docker

#systemctl status docker



## **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 ./

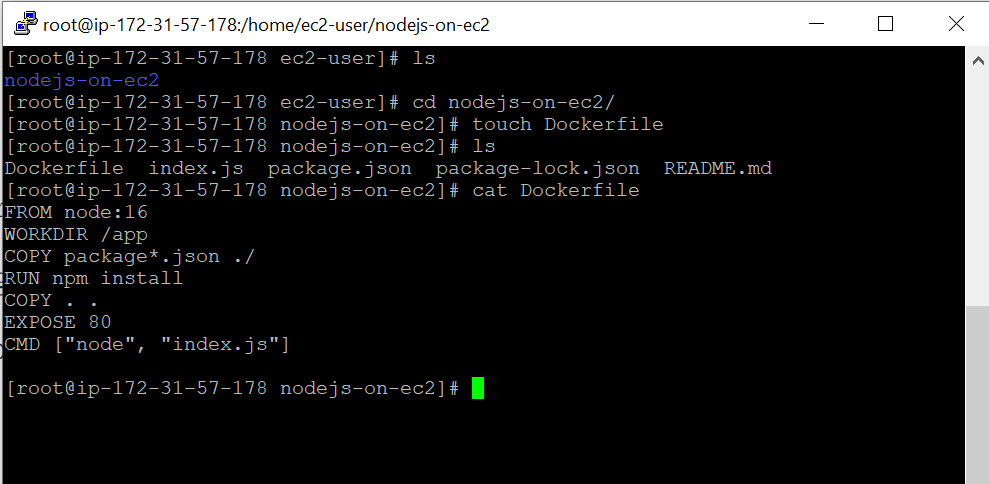
RUN npm install

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.

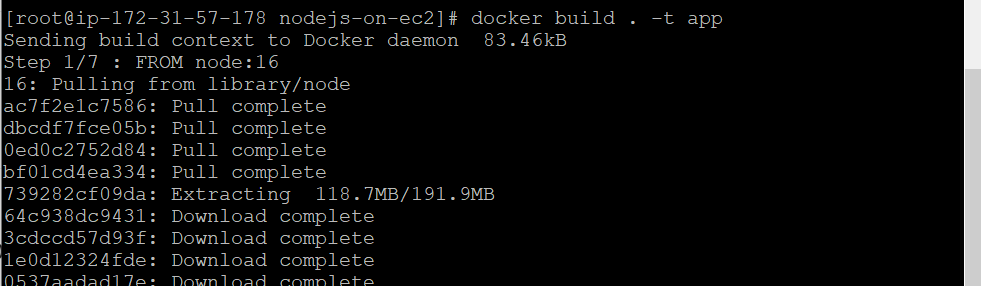
Use command :

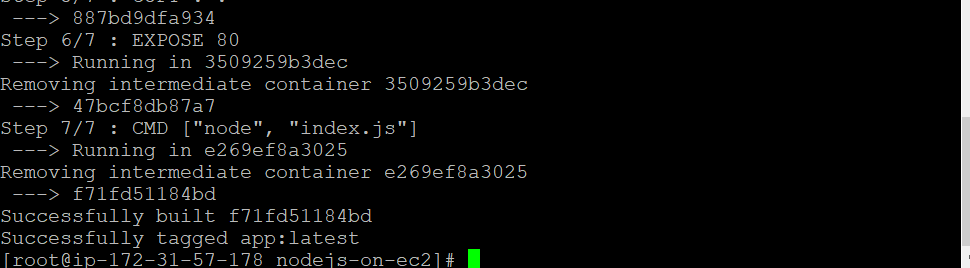
#docker ps

#docker ps -a

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

# docker build . -t app





## **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.

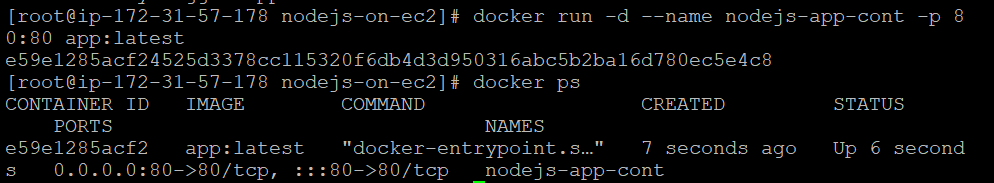
**EXTRA**

**docker run -p 8080:80 your\_image\_name**

In this example, it maps port 80 from the container to port 8080 on the host. Adjust the host port as needed.

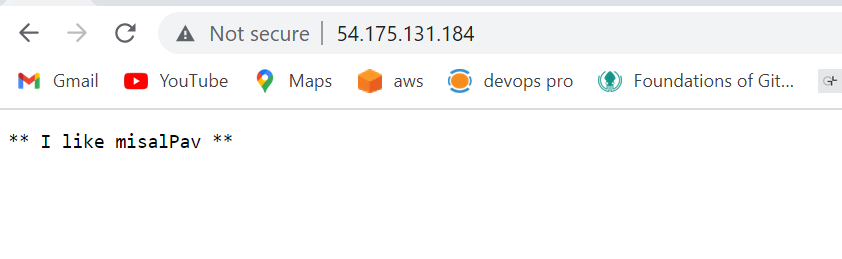
when using Docker, the host port and the container port can be different. The -p (or --publish) option allows you to specify the mapping between a port on the host machine and a port on the Docker container.

**docker run -p <host\_port>:<container\_port> your\_image\_name**



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.🌻✨

**NOTE : If not accessible make sure you have exposed the port 80 in the Security of the server.**

**FINAL COMMANDS FOR HANDSON**

#yum install git -y

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

#yum install docker -y

#docker -v

#systemctl start docker

#systemctl status docker

DF

#docker ps

#docker ps -a

#docker build -t rajjo103/app .

# docker run -d --name nodejs-app-cont -p 80:80 rajjo103/app:latest

docker login

docker push rajjo103/app:latest

**What is Detached Mode in Docker?**

In Docker, detached mode is a way to run containers in the background, allowing you to continue using the terminal or shell without being attached to the container's console. This is useful when you want a container to run in the background without blocking your terminal.

To run a Docker container in detached mode, you can use the -d option with the docker run command. Here's a basic example:

docker run -d IMAGE\_NAME