1. Create an ec2 instance with http, ssh and allow security groups
2. Connect to the ec2 instance using “**ssh -i xyz.pem ec2-user@public-ip”**
3. Install docker and add user to docker group using the following commands

* sudo yum update -y
* sudo yum install docker -y
* sudo service docker start
* sudo systemctl enable docker
* sudo usermod -aG docker ec2-user 🡪logout of the instance and login back again to add the user to docker group inorder to get access to the docker commands

1. Install nginx into ec2 instance

* docker pull nginx: version 🡪 pulls the particular version mentioned. If no version specified then it will pull the latest version of nginx from docker hub
* docker images 🡪 to check the image is created or not

image will be created and now we need to create a container for the above image

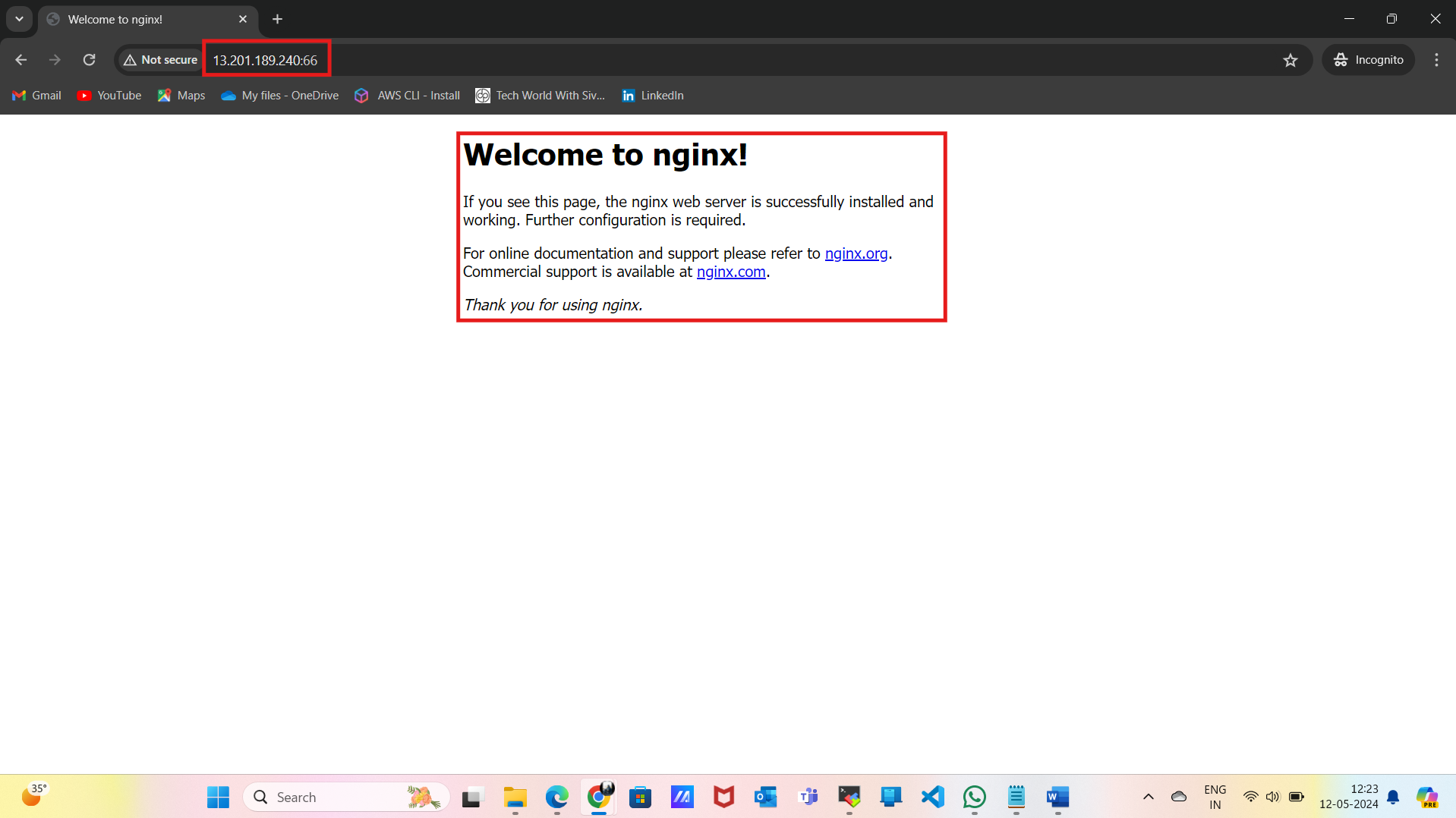
* docker create <**image id**> 🡪 create a container from the image
* docker ps -a 🡪 to check the container is created or not

container will be created

1. docker commands

* docker ps 🡪 to get the list of running containers in the server
* docker ps -a 🡪 to get the list of all containers in the server
* docker images 🡪 to get the list of images in the server
* docker - -version 🡪 to get the version of docker
* docker start <**container id**> 🡪 to start the container
* docker inspect <**container id**> 🡪 to get the complete info about the container
* **docker run image: version 🡪 docker pull + docker create + docker start**
* docker stop <**container id**> 🡪 to stop the running container
* docker rm <**container id**> 🡪 to remove the container from server
* docker rm -f $(docker ps -aq) 🡪 to remove all the containers at once
* docker rmi <**image id**> 🡪 to remove the docker image from the server
* docker rmi -f $(docker images -aq) 🡪 to remove all the images at once (f stands for removing the image or container forcefully)
* docker run -d image: version 🡪 by default the container will run in the foreground, by using -d (detached mode) the container runs in background
* docker run -d -p 66:80 nginx: latest 🡪 exposing the port to outside world. here container port [80] is exposed to host port [66]. By using host port [66], we can access through outside world (**INTERNET**) **{public-Ip: host port}**

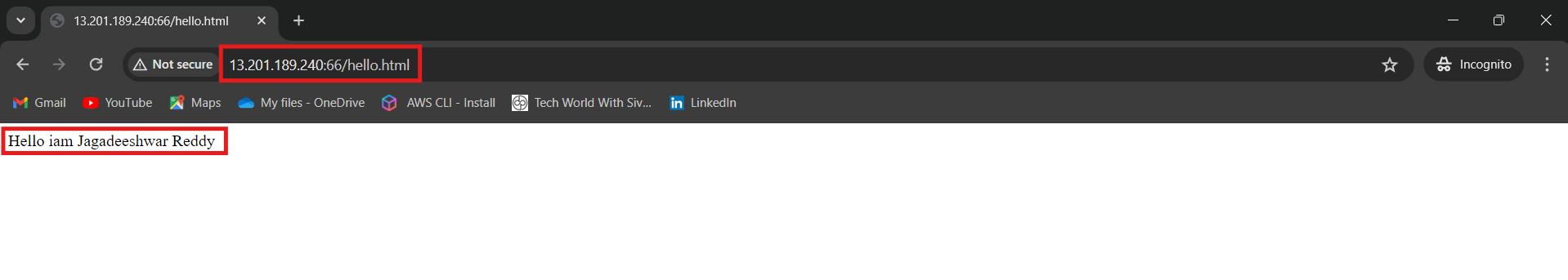
1. container is the running version of image
2. Image is a static single file; it can run any number of containers.



* docker exec -it < **container id** >bash 🡪 to enter inside the container. -it is interactive terminal
* CTRL + D 🡪 to come out of the container
* docker exec cat /etc/\*release 🡪 will give us the underlying OS.
* cd **usr/share/nginx/html/** 🡪 The default html location for nginx

EX: - echo “Hello i am Jagadeeshwar Reddy" > hello.html

we can access through (**INTERNET**) using **{public-Ip: host port/hello.html}**



|  |  |  |  |
| --- | --- | --- | --- |
|  | **PHYSICAL MACHINE** | **VIRTUAL MACHINE** | **CONTAINERS** |
| SCALABILITY | BAD | AVERAGE | GOOD |
| RESOURCE OPTIMISATION | BAD | AVERAGE | GOOD |
| HIGH AVAILABILITY | BAD | AVERAGE | GOOD |
| COST | BAD | AVERAGE | GOOD |
| PROVISIONING | BAD | AVERAGE | GOOD |
| PERFORMANCE | GOOD | AVERAGE | GOOD |
| SECURITY | GOOD | AVERAGE | BAD |
| PORTABILITY | BAD | AVERAGE | GOOD |
| CONSISTENT ENV | BAD | AVERAGE | GOOD |

* **dockerfile: -** Declarative approach to create our own images

**To clone and build git in ec2 instance**:

* **sudo yum install git -y**
* **git clone https://github.com/Jagadeeshwar123/Dockerfiles.git <git repo link>**
* **cd dockerfiles <local directory>**
* **cd FROM**
* **ls -l**

DOCKERFILE

**FROM: -** FROM is the first instruction in dockerfile. It refers to the base OS, to create an image we always refer to the base OS

FROM <base OS>:<version>

* docker build -t **<image-name>:<version>** . ( . refers to the current directory, intimating the docker engine that the docker file is in current directory) [from local]
* docker build -t **<Docker-hub-url>/**username/image-name:version . (from docker hub)
* docker run **name:version 🡪** run command creates the container from image
* docker run **name** sleep 5 🡪 it will install and sleep for 5 seconds

**RUN: -** If we need to install or configure something on top of the base OS then we need to use run command

RUN yum install nginx -y

**CMD: -** It is the instruction to run the container. CMD should always run in foreground and it should run infinite times. If it is not running in the foreground then the container will exit

CMD [EXECUTABLES]

**COPY: -** COPY instruction is to copy the files from local to image

COPY <local path> <path inside the image>

**RUN VS CMD: -**

**RUN instruction will be executed at the time of image creation**

**CMD instruction will be executed at the time of container creation**

**# base OS**

**FROM almalinux:8**

**# installing or updating the required packages**

**RUN yum update -y**

**# installing web server**

**RUN yum install nginx -y**

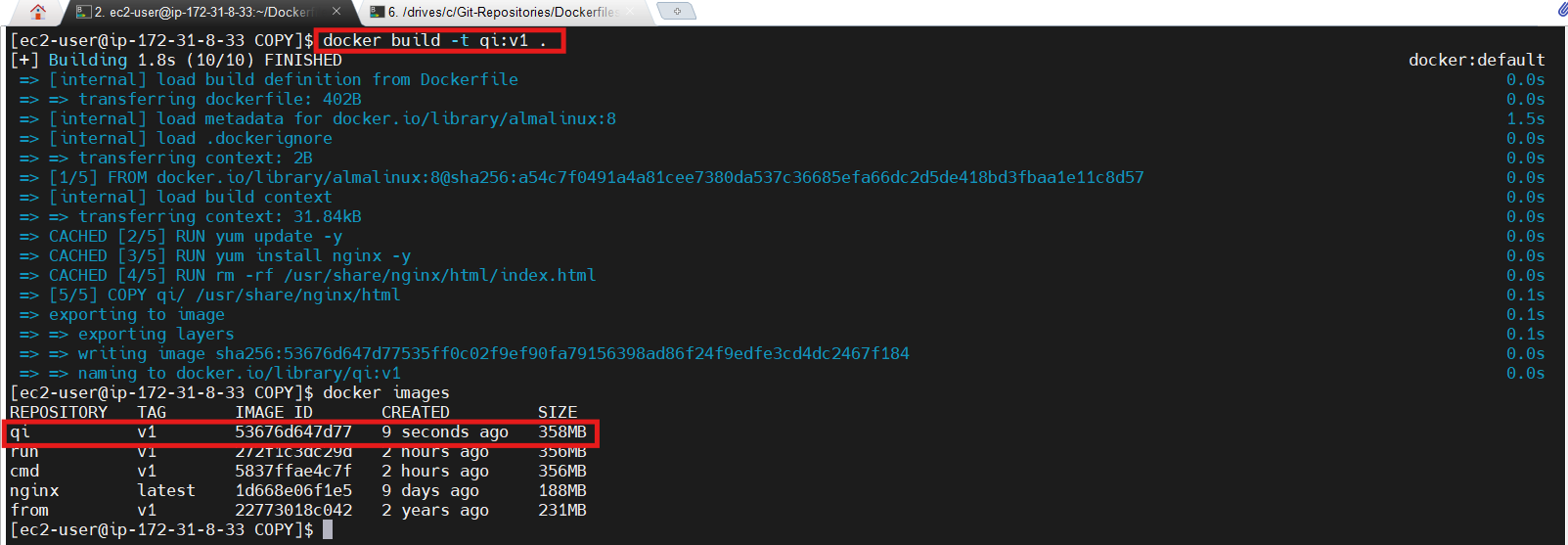
**RUN rm -rf /usr/share/nginx/html/index.html**

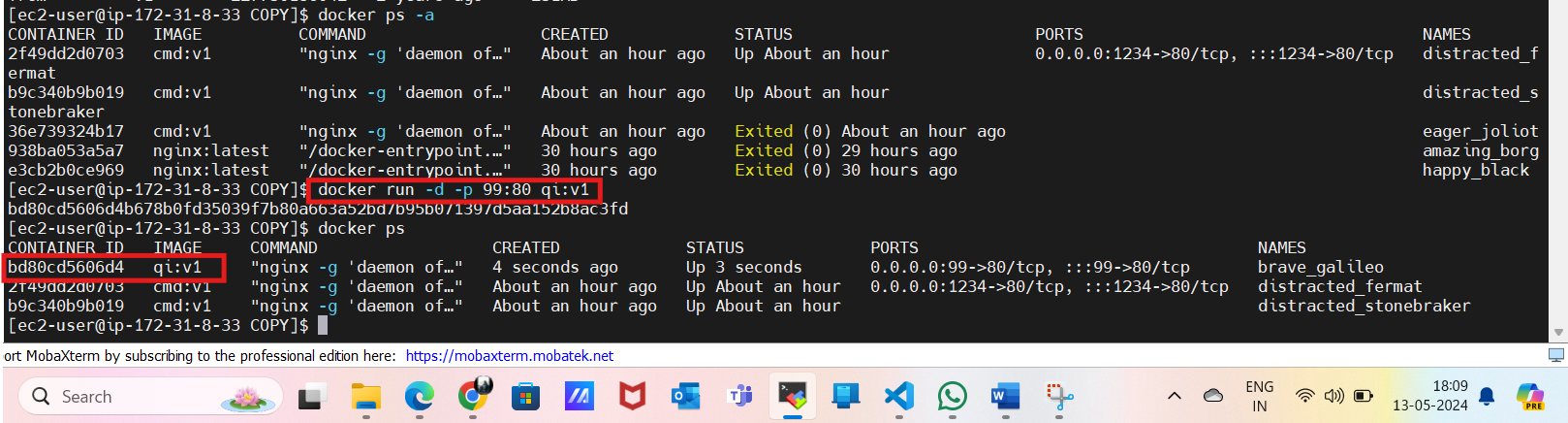
**# copying app code**

**COPY qi/ /usr/share/nginx/html**

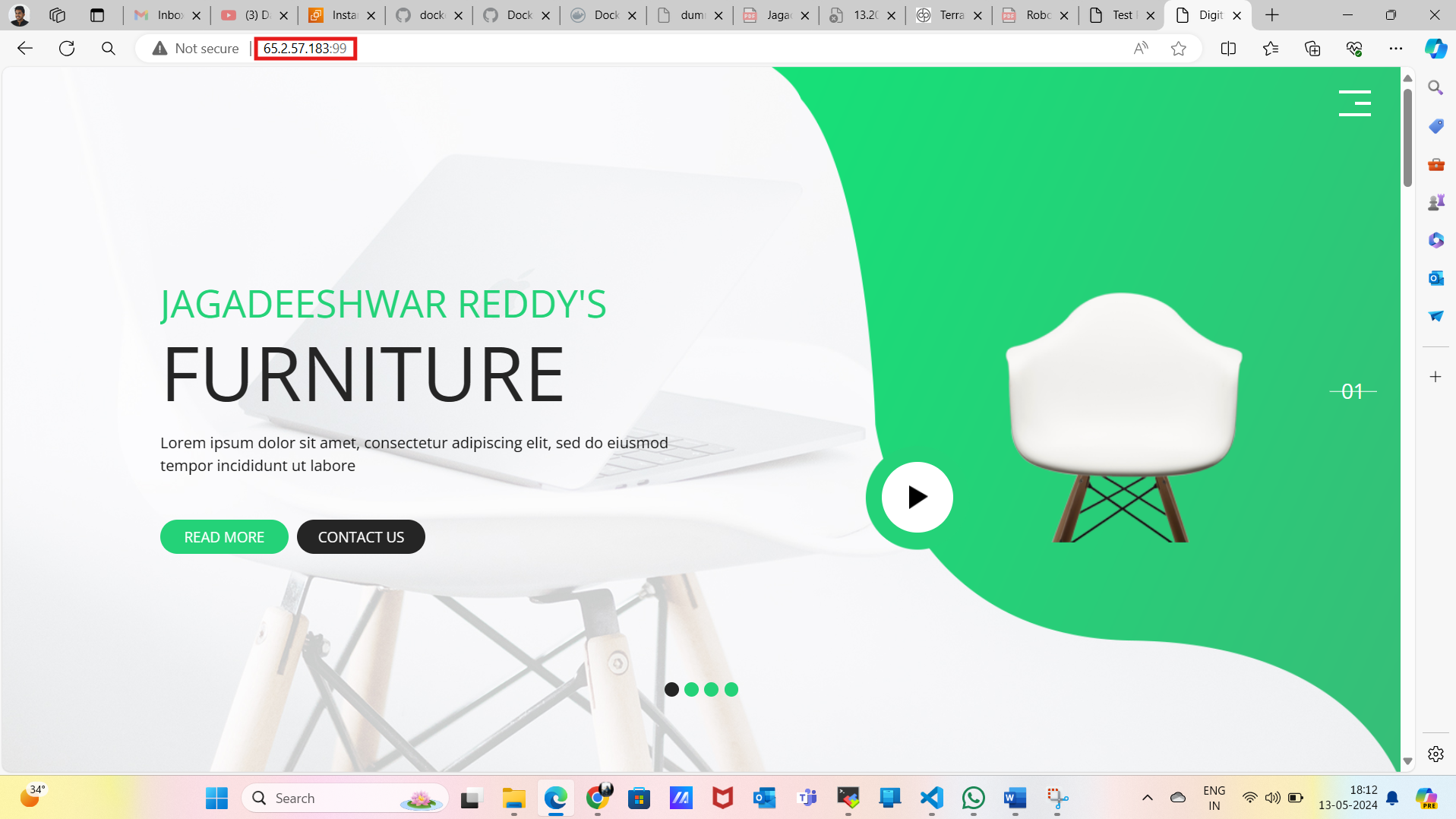
**# starting the server**

**CMD [ "nginx", "-g", "daemon off ;" ]**





* Deployed a website by using copy command in the container
* We can also deploy using the domain instead of Ip address
* Navigate to Route 53 in aws and click on hosted zone give a domain name to the route “**Jagadeesh.com”**
* Click on create record and Paste the Ip address in the route and save
* Now if we hit “**Jagadeesh.com”** we can see the website on the web browser
* For this we need to have a personal domain “**Jagadeesh.com”** already.



**LABEL: -** It is used to filter the images among many

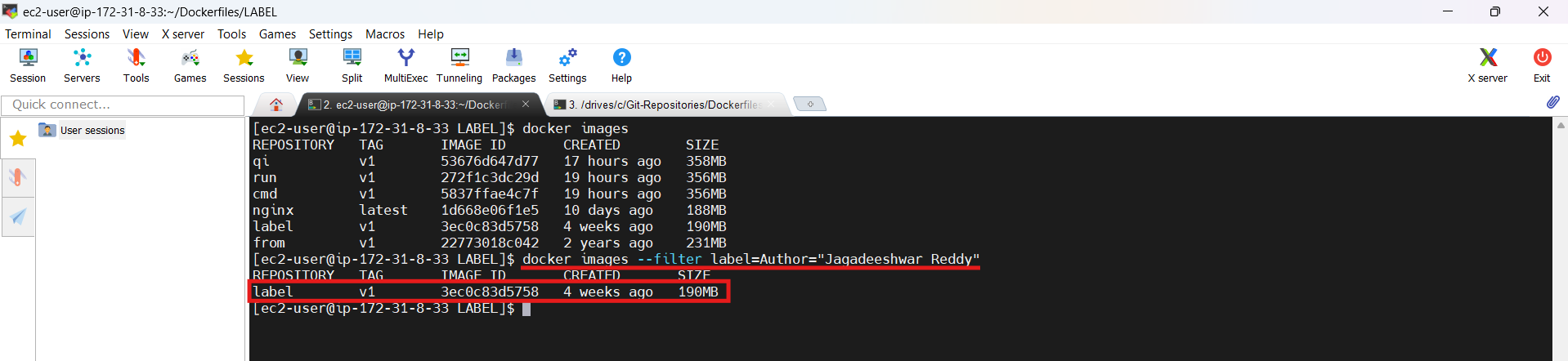
**FROM almalinux:8**

**LABEL Author="Jagadeeshwar Reddy" \**

**company="Broadridge" \**

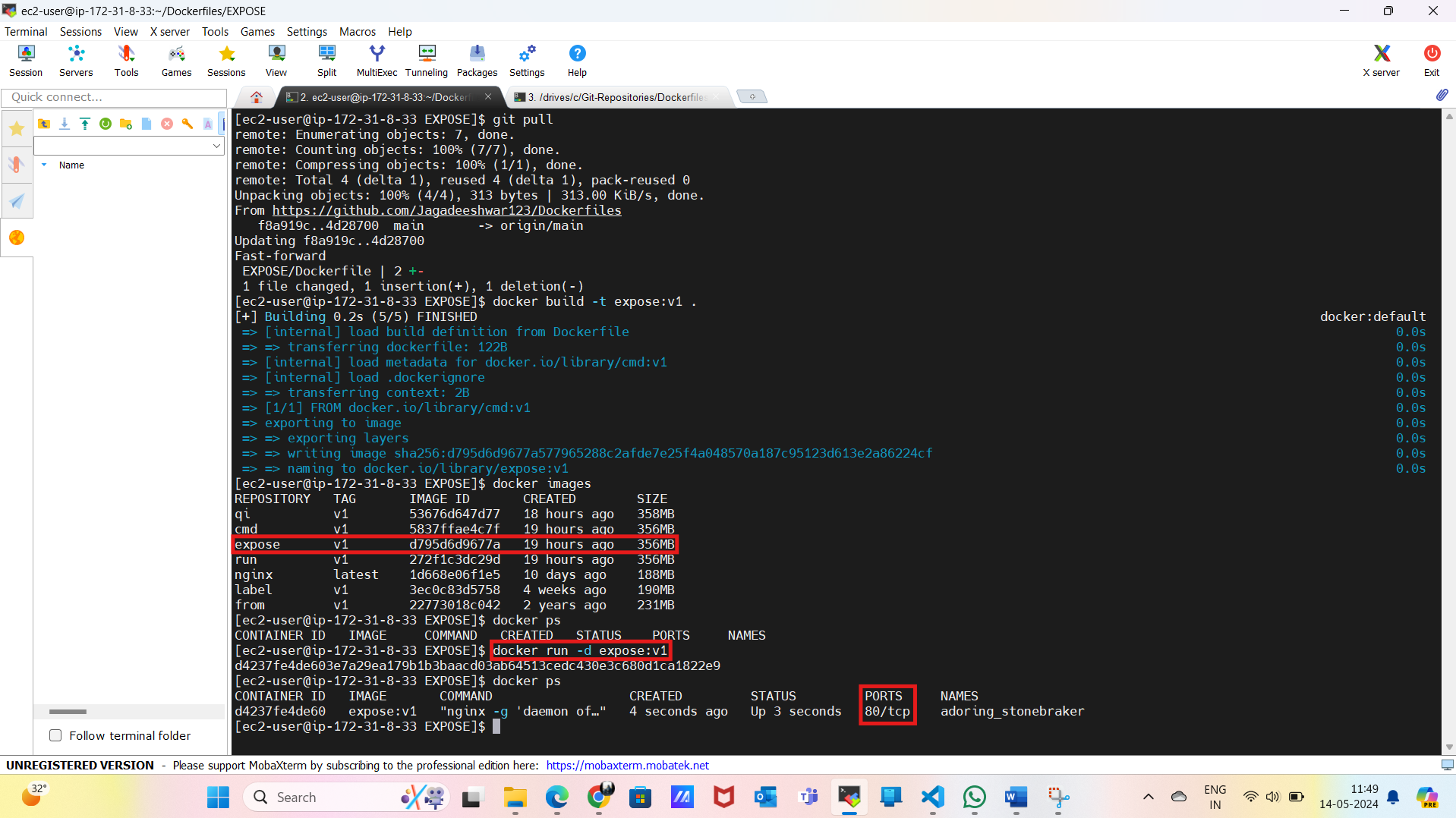
**Role="DevOps Engineer"**

* By adding the labels, we can filter images among 100’s or 1000’s of images. Using **{docker images --filter label=key** **(**Company**) =value (**Broadridge**)}**



**EXPOSE: -** It is used to know which port and the protocol used in the image creation

EXPOSE <port/protocol>



**ENV: -** It is used to set env variables **(APP CONFIG)** using key value pairs

**FROM almalinux:8**

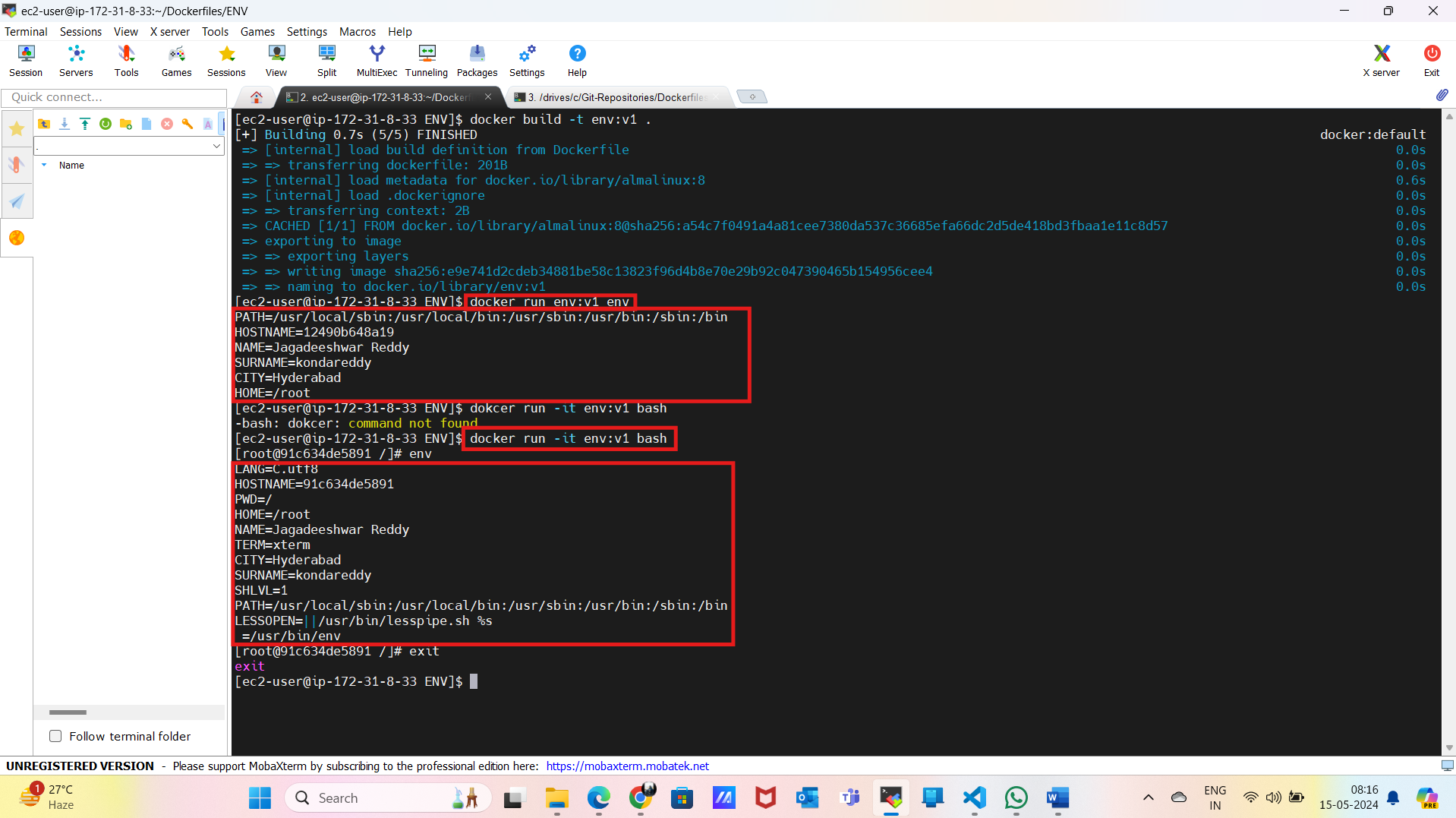
**ENV NAME="Jagadeeshwar Reddy" \**

**SURNAME="kondareddy" \**

**CITY="Hyderabad"**

We can get either using **{docker run env:v1 env or docker run -it env:v1 bash** and **env}**

We can override the values using **{docker run -e CITY=Banglore env:v1 env}**



**ADD: -**

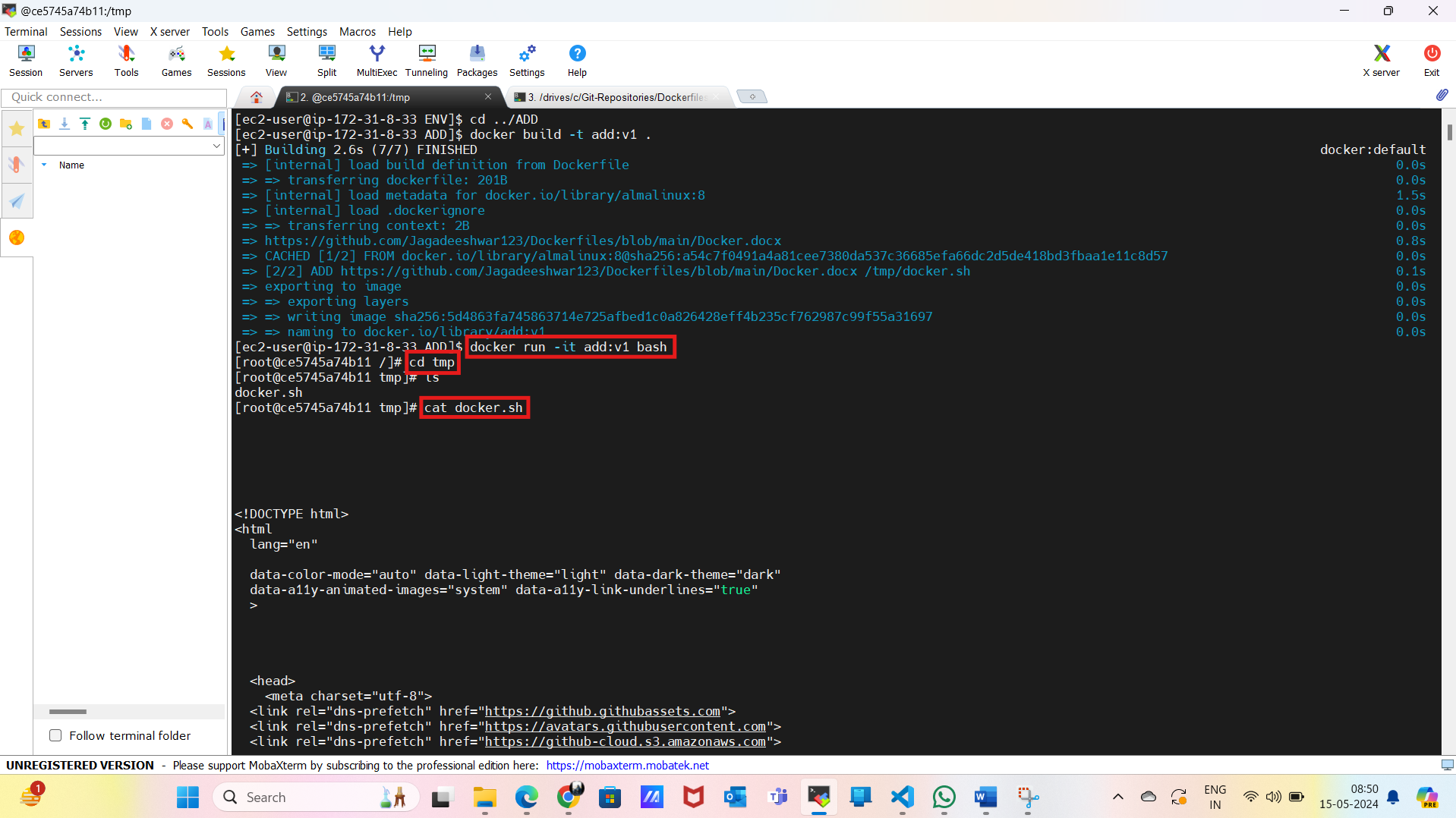
* It is as same as **COPY** but **ADD** has some extra capability to download the content from internet
* It can untar/unzip the files and copy the content to the image from tar file to image

ADD <source> <dest>

**FROM almalinux:8**

**ADD https://github.com/Jagadeeshwar123/Dockerfiles/blob/main/Docker.docx /tmp/docker.sh**

**ADD sample.tar /tmp**

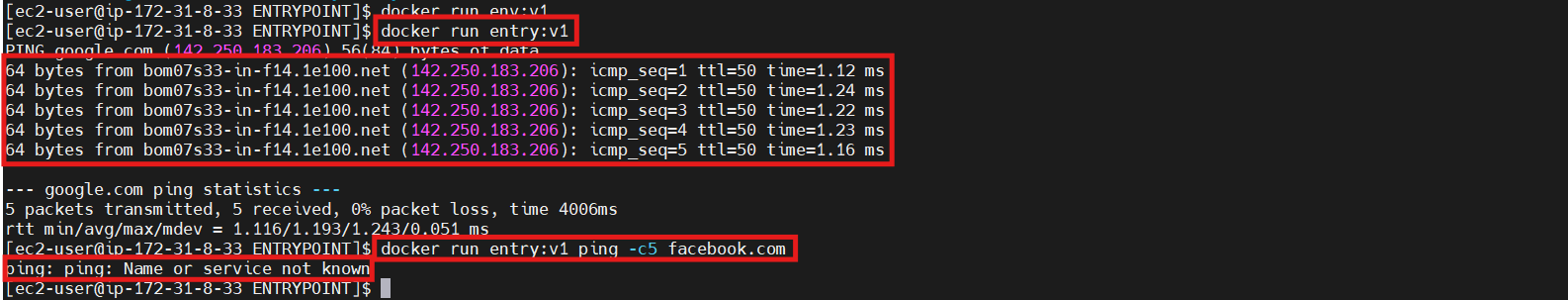


**ENTRYPOINT: -**

* **ENTRYPOINT** is as same as **CMD** but CMD can be overridden whereas ENTRYPOINT can’t be overridden, instead it will append to the command
* If we use both CMD and ENTRYPOINT, CMD will provide arguments to ENTRYPOINT
* CMD is used to provide default arguments to the ENTRYPOINT, but we can always override CMD from command line

**FROM almalinux:8**

**ENTRYPOINT [ "ping", "-c5", "google.com" ]**



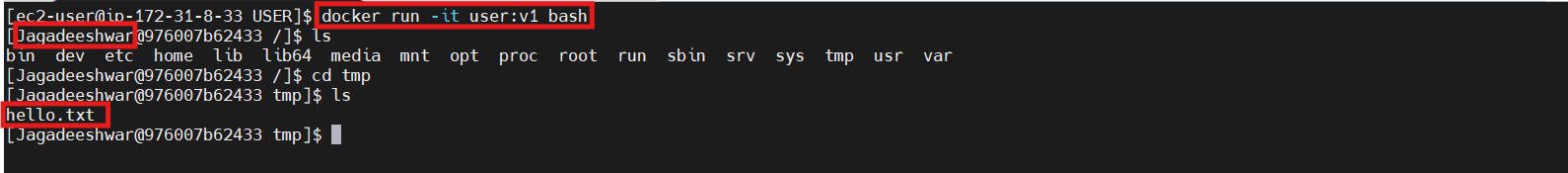
**USER: -** It is useful to restrict the root privileges inside the image and container

**FROM almalinux:8**

**RUN useradd Jagadeeshwar**

**USER Jagadeeshwar**

**RUN echo "Hello" > /tmp/hello.txt**

****

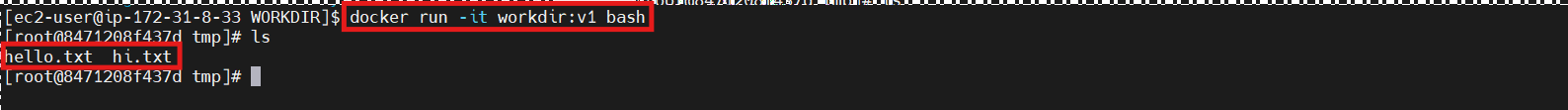
**WORKDIR: -** It is used to set the path to our docker images and container. After **WORKDIR** commandall the run commands will fall inside **WORKDIR**

**FROM almalinux:8**

**WORKDIR /tmp**

**RUN touch hello.txt**

**RUN touch hi.txt**

****

**ARG: -**

* Every time **FROM** should be the first instruction but **ARG** is the only instruction used before **FROM**
* **ENV** variables will be accessed both in image and container creation. Whereas **ARG** is only accessible at the time of image creation
* We can give the value from the dockerfile or we can give it from command line
* If we want to access the **ARG** in container also, then create one env variable and assign the value of ARG to that. Then we can access **ARG** values through **ENV** both in image and container.

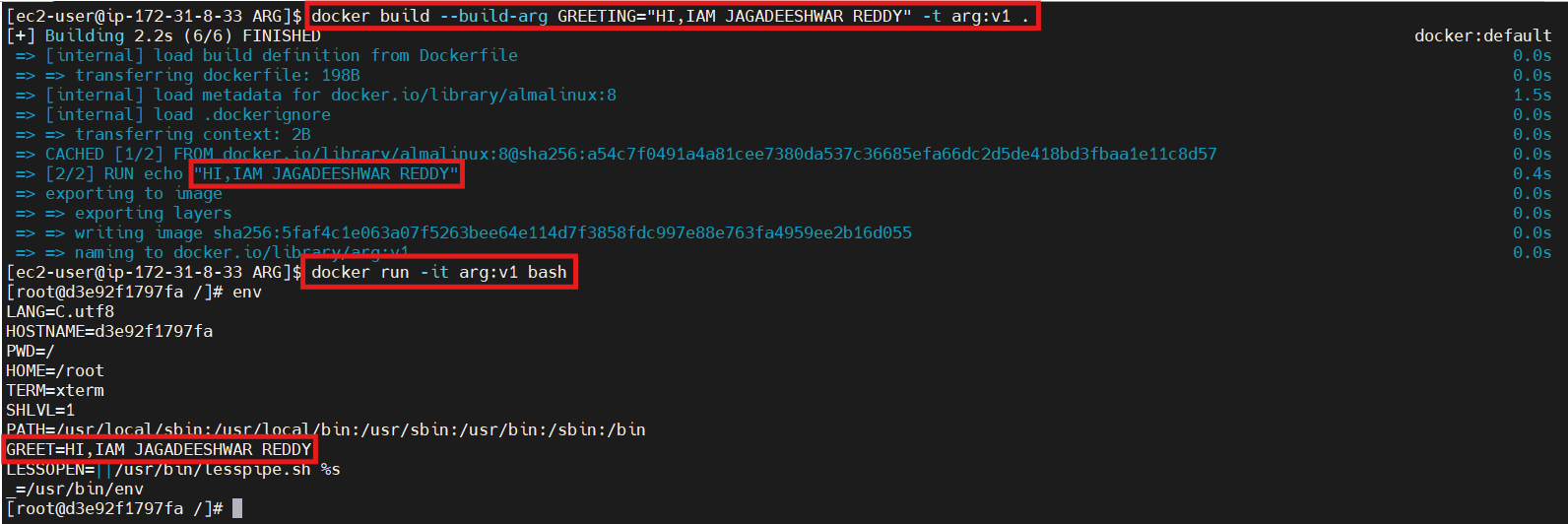
**ARG VERSION**

**FROM almalinux:${VERSION:-8}**

**ARG GREETING="HI, I AM JAGADEESHWAR REDDY"**

**ENV GREET=${GREETING}**

**RUN echo "${GREETING}"**

**ONBUILD: -**ONBUILD is used to set some hard guidelines to the image. We can control how others can use our image as their base image

**FROM almalinux:8**

**RUN yum update -y**

**RUN yum install nginx -y**

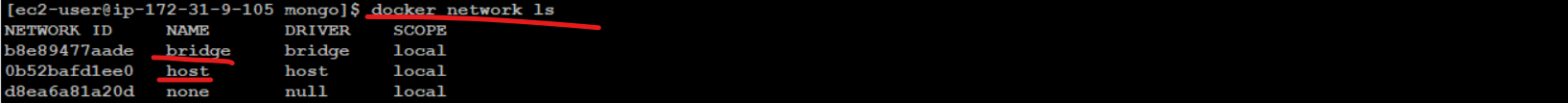
**RUN rm -rf /usr/share/nginx/html/index.html**

**ONBUILD ADD index.html /usr/share/nginx/html/index.html**

**ENTRYPOINT [ "nginx", "-g", "deamon off;" ]**

**NETWORKING: -**

* In docker we have two types of networks **HOST** and **BRIDGE.** By default, we use bridge network

****

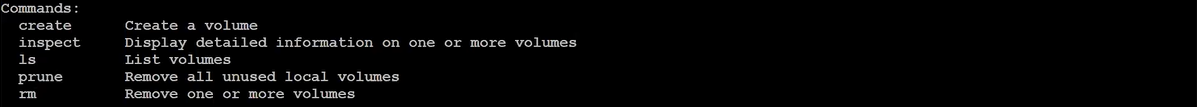
* 2 power 16 Ip addresses possible for default network **(docker 0)**
* We cannot connect containers with names using default network
* --network is a network mapping as **(--network name)**

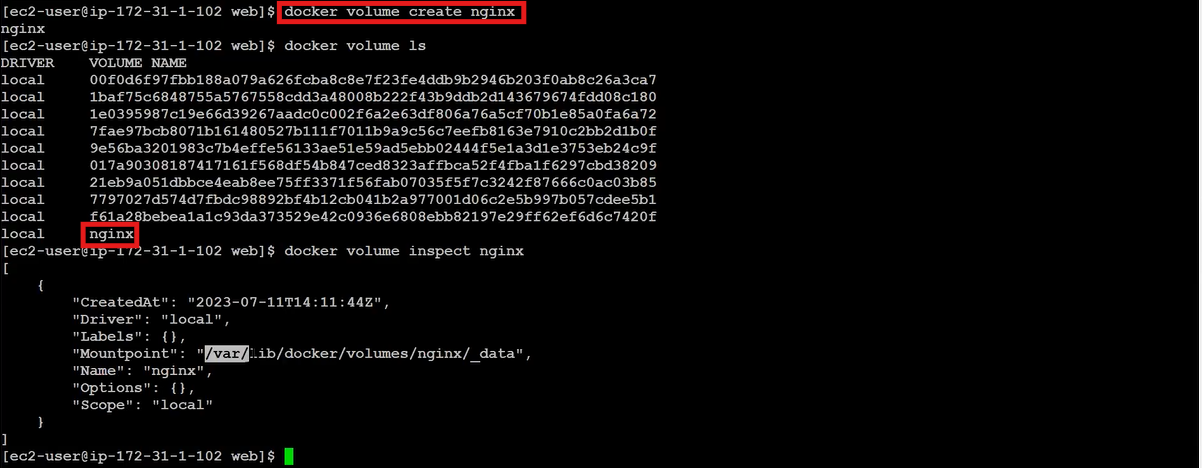


**VOLUMES: -**

* Containers are ephemeral in nature; it will not store the data by default, once container is deleted, we lose the data
* If we don’t want to lose the data , create our own volumes
* -v is a volume mapping as **(-v volumename:path inside the container)**

**ex:- “-v nginx:/usr/share/nginx/html “.** we’ll get the path from Docker hub

****

****

* Named volumes are managed by docker. Un-named volumes are not managed by docker
* At **/var/lib/docker/volumes/**mongodb in root **(sudo)** user we can see the data present in that particular volume

**DOCKER COMPOSE: -**

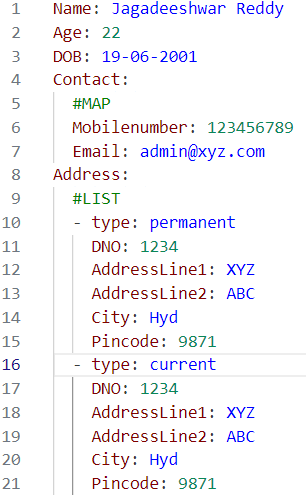
**XML: -** Extensive Markup Language



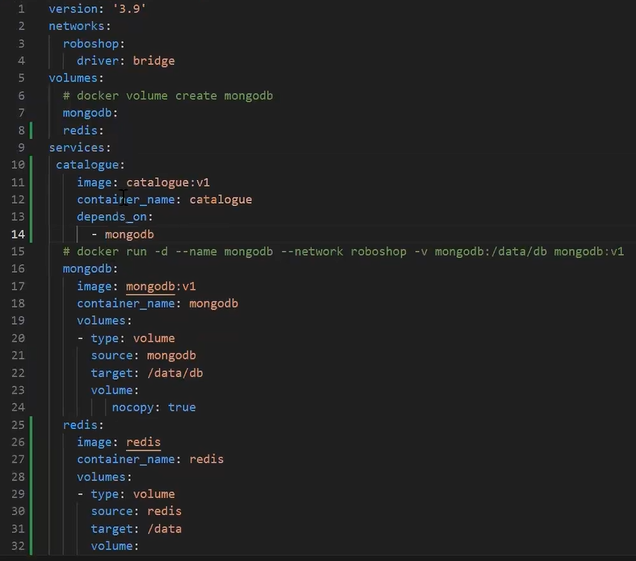
**JSON: -** JavaScript object notation

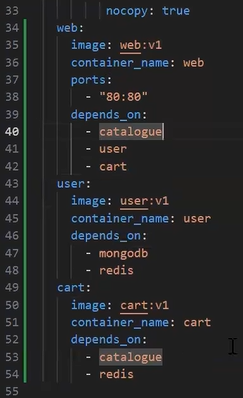
****

**YAML: -** Yet another markup language

****

* Docker compose is a declarative way of running the images as containers with dependencies
* We can start and stop all the containers at once
* We can create dependencies between containers
* **docker-compose up -d 🡪** Run the docker compose to create the containers
* **docker-compose down 🡪** To stop and to remove the containers

****

****

**Docker Best Practices: -**

[Top 8 Docker Best Practices for using Docker in Production ✅ - DEV Community](https://dev.to/techworld_with_nana/top-8-docker-best-practices-for-using-docker-in-production-1m39)

1. **Use Official Base Images**
2. **Keep Images Small**
3. **Optimize Dockerfile Instructions**
4. **Security Best Practices**
5. **Use .dockerignore**
6. **Limit Container Capabilities**
7. **Container Orchestration**
8. **Use Environment Variables**
9. **Health Checks**
10. **Logging and Monitoring**
11. **Automate Builds**
12. **Versioning and Tagging**
13. **Don’t run containers with root privileges**

* Always create a system user and finally our image should be on system user

**RUN addgroup -S cart && adduser -S cart cart**

**WORKDIR opt/server**

**RUN chown -R cart:cart /opt/server**

**USER cart**

**Docker layers: -** keep the static instructions at the top and frequently used instructions at below

* We can use run command as **RUN yum install git -y \ && yum install maven -y**

**Multi Stage Docker Builds: -**

* Alpine is the bare minimum OS which has very less memory space.
* Multistage build is a technique of reducing the image size by getting the output into an image. Use first docker file as builder and copy the output of first docker file to the second docker file.
* First docker file should contain **AS builder** inorder to use it for the second docker file.

**Disadvantages of Docker: -**

1. Docker host crashes | HIGH AVAUILABILITY
2. Increase in sudden traffic like million requests | SCALABILITY
3. Eco system
4. Where to store secrets
5. Environment Variables
6. Load Balancing. How to balance the load between similar containers

For all these disadvantages we use **Container Orchestration tool i.e. “Kubernetes”**