**SESSION - 52**

* we discussed legacy,discussed mono lithic, discussed micro services and bare metal fetch, VM vs containers.
* docker ps -a -- if you want to see all containers in any state.
* docker images - display the images ( available in server).
* docker pill - pull the image from the docker hub.
* docker create nginx -- create the container out of the image.
* docker start - start the container.
* Instead of pull create we can run docker.
* Run at a time **docker run -d -p 80:80 nginx**
* -d : detach mode, to send the container in the background. Instead of blocking the screen,
* -p we are added, hostport with container port for coupling actually that is port forwarding. Because container will run sever inside that’s why they will not access directly containers so first traffic should be reached to host.
* We are using host port:container port
* Host port you can nit reuse. Container port you can because every container have 0 - 65535 ports.
* Then for login exact -it (interactive terminal), your container name or your container id and bash just get the terminal.

**docker exec -it nginx bash**

* Docker inspect - you can see all the information about the container. You can see the ip address and everything.
* We itself have to build the images we have way that is called docker file.

**Docker file?**

Docker file is set of docker instructions you used build the custom images.

* We saw **FROM** instruction -- From should be the first instruction inside docker file represents the base OS you use.

**docker build -t from:v1**

Docker build - how to build the docker images.

-t -- giving the tag

Form:v1 -- images name colon and version

Docker presents current directory you should have file name called Dockerfile.

* **Run**

Run is used at the time of image building you can run in instruct to configure the images and installing the packages etc.

Docker file can have multiple run executions,

* **CMD** run execution at the time of container creation. Dockerfile we have only one cmd instruction that is the difference between that.
* **COPY and ADD --** copy and add to both are used to copy the content from the local directory into the directory but **ADD** we have two more advantages

1. You can directly download the file from internet to the image.
2. it can directly untar the file into container.

**Creating instance through docker**

**Create a Docker directory**

**File data.tf**

data "aws\_ami" "joindevops" {

owners = ["973714476881"]

most\_recent = true

filter {

name = "name"

values = ["RHEL-9-DevOps-Practice"]

}

filter {

name = "root-device-type"

values = ["ebs"]

}

filter {

name = "virtualization-type"

values = ["hvm"]

}

}

**docker.sh**

#!/bin/bash

dnf -y install dnf-plugins-core

dnf config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo

dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin -y

systemctl start docker

systemctl enable docker

usermod -aG docker ec2-user

growpart /dev/nvme0n1 4

lvextend -L +20G /dev/RootVG/rootVol

lvextend -L +10G /dev/RootVG/varVol

xfs\_growfs /

xfs\_growfs /var

**docker.tf**

resource "aws\_instance" "docker" {

ami = local.ami\_id

instance\_type = "t3.micro"

vpc\_security\_group\_ids = [aws\_security\_group.allow\_all\_docker.id]

# need more for terraform

root\_block\_device {

volume\_size = 50

volume\_type = "gp3" # or "gp2", depending on your preference

}

user\_data = file("docker.sh")

#iam\_instance\_profile = "TerraformAdmin"

tags = {

Name = "${var.project}-${var.environment}-docker"

}

}

resource "aws\_security\_group" "allow\_all\_docker" {

name = "allow\_all\_docker"

description = "allow all traffic"

ingress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

ipv6\_cidr\_blocks = ["::/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

ipv6\_cidr\_blocks = ["::/0"]

}

lifecycle {

create\_before\_destroy = true

}

tags = {

Name = "allow-all-docker"

}

}

**locals.tf**

locals{

ami\_id = data.aws\_ami.joindevops.id

}

**provider.tf**

terraform {

required\_providers {

aws = {

source = "hashicorp/aws"

version = "5.98.0"

}

}

}

provider "aws" {

# Configuration options

region = "us-east-1"

}

**variables.tf**

variable "project" {

default = "roboshop"

}

variable "environment" {

default = "dev"

}

**.gitignore**

# Local .terraform directories

\*\*/.terraform/\*

# .tfstate files

\*.tfstate

\*.tfstate.\*

# Crash log files

crash.log

# Exclude all .tfvars files, which are likely to contain sentitive data, such as

# password, private keys, and other secrets. These should not be part of version

# control as they are data points which are potentially sensitive and subject

# to change depending on the environment.

#

.tfvars

# Ignore override files as they are usually used to override resources locally and so

# are not checked in

override.tf

override.tf.json

\*\_override.tf

\*\_override.tf.json

# Include override files you do wish to add to version control using negated pattern

#

# !example\_override.tf

# Include tfplan files to ignore the plan output of command: terraform plan -out=tfplan

# example: \*tfplan\*

# Ignore CLI configuration files

.terraformrc

terraform.rc

# Ignore terraform locks

.terraform.lock.hcl

# Ignore terragrunt cache

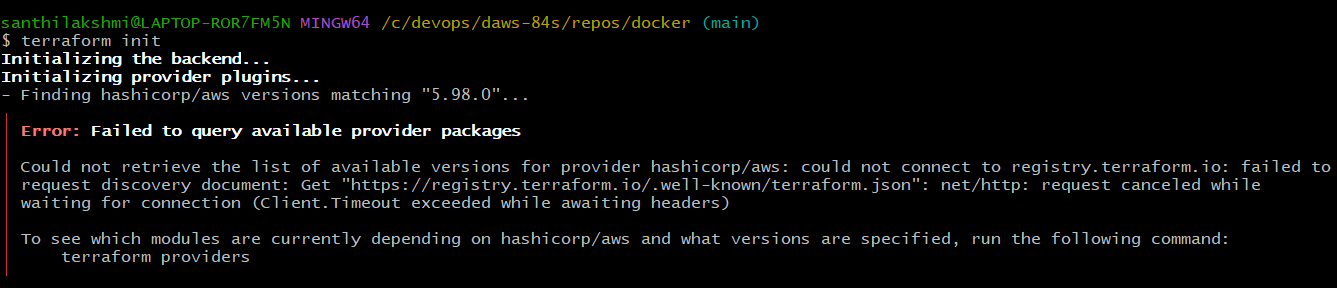
.terragrunt-cache

testing/

* In docker.tf security group allowing every port.
* In docker.dh -- docker file install,start,enable and to docker group added ec2 user group.
* 15gb root storage -- 10 0r 20 gb giving to volume.
* Docker Home diectory if you ask me - /var/lib/docker -- if increasing images diectory size also will increase. So increase the /var/memory. So 12gb was increased you can increase more if you need.

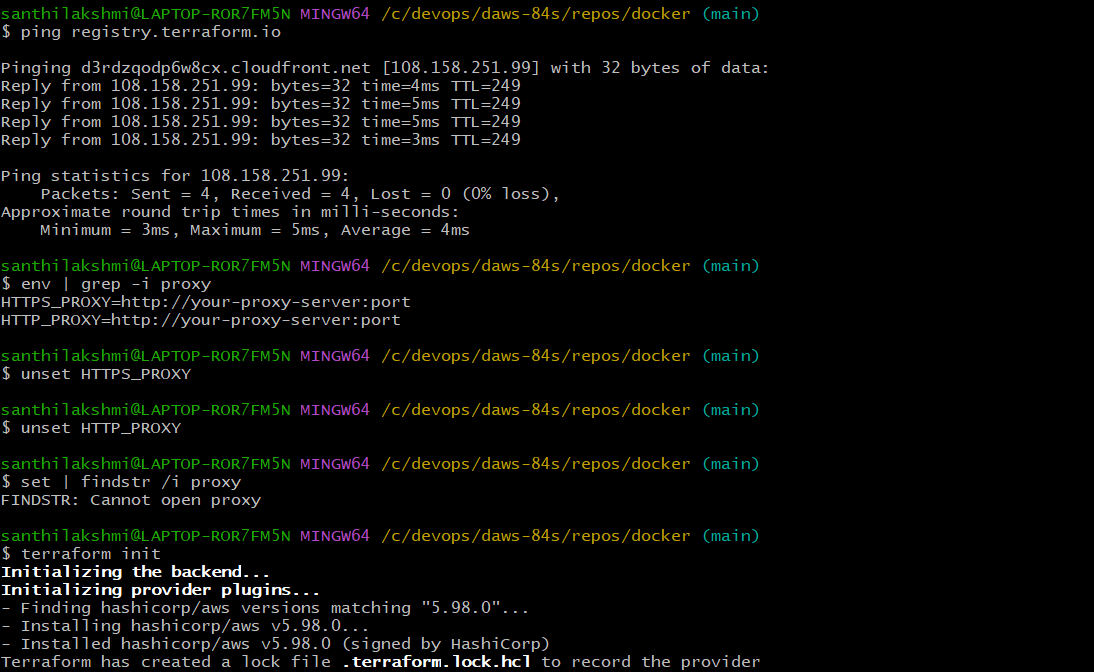
Dockerfile reference -- search in google

terraform init

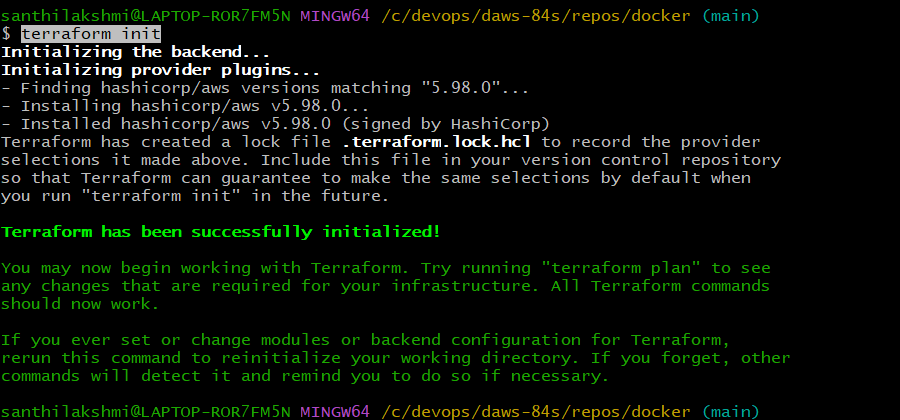


I got this error

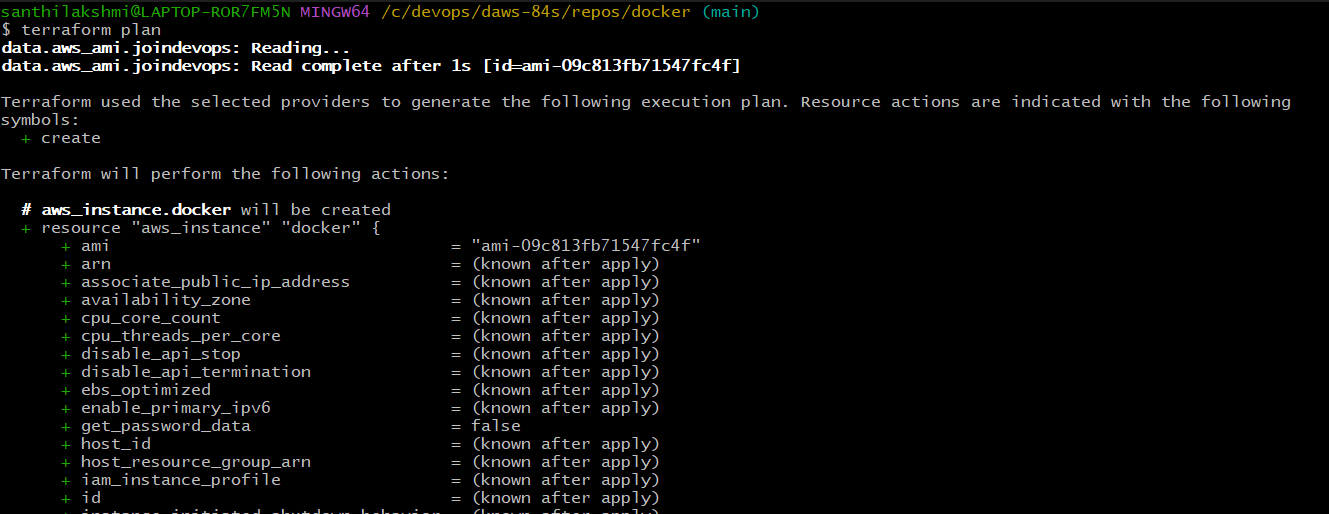
Please check the below snap for rectifying the error



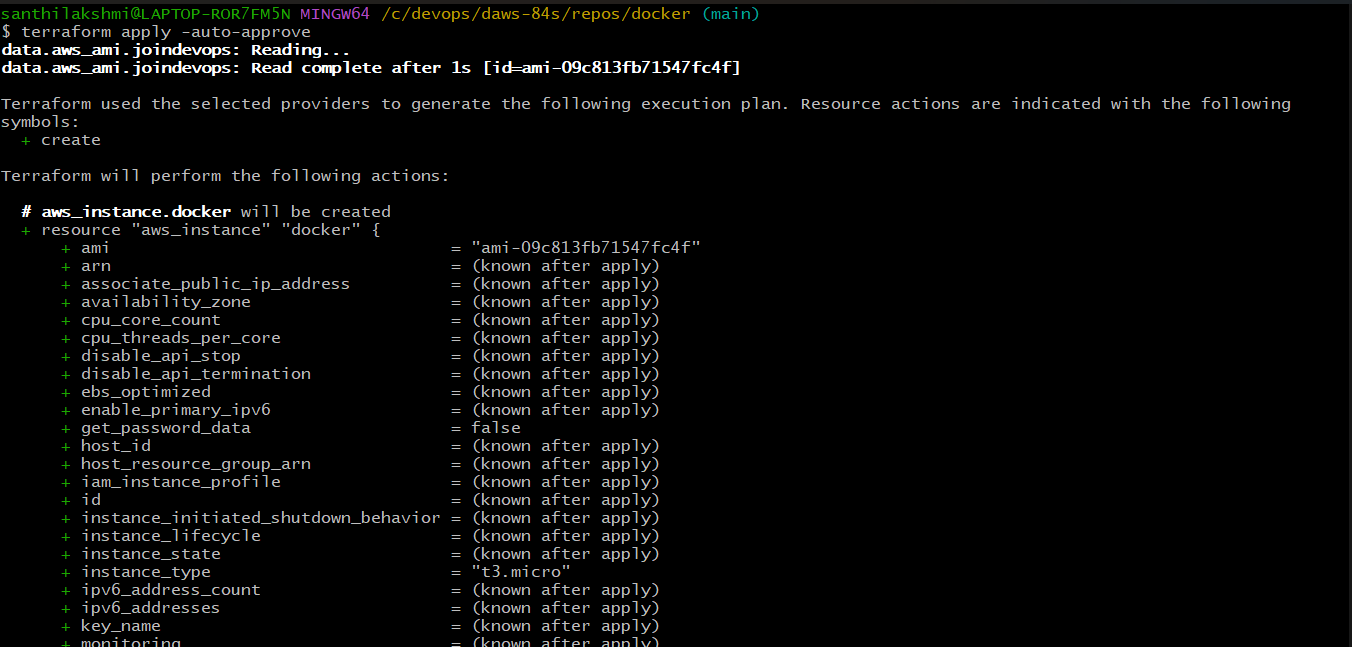
terraform init



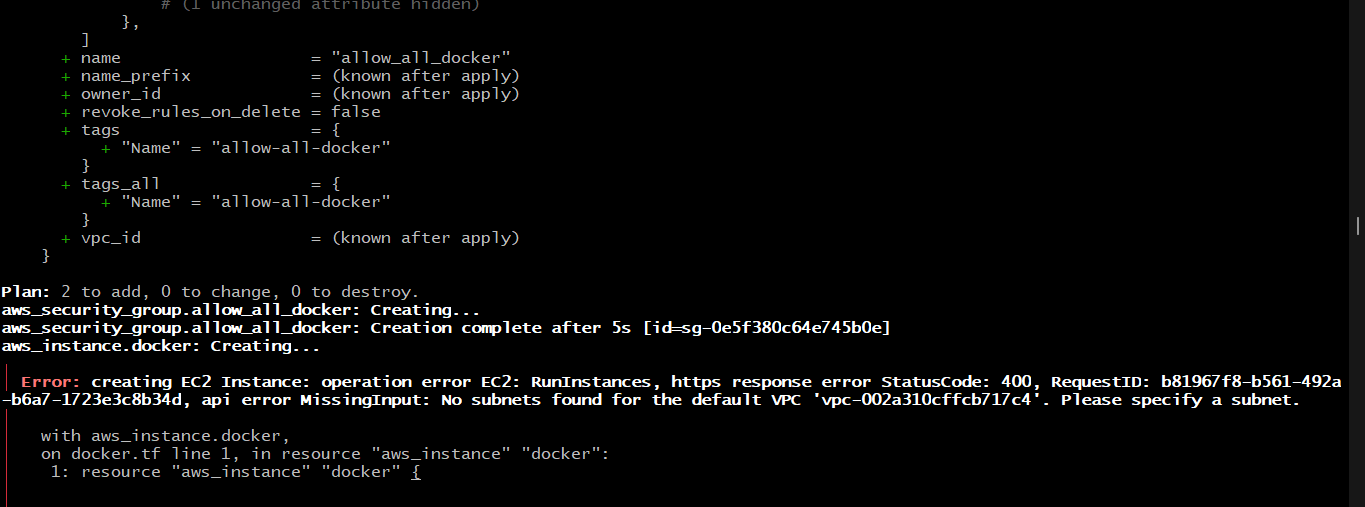
terraform plan



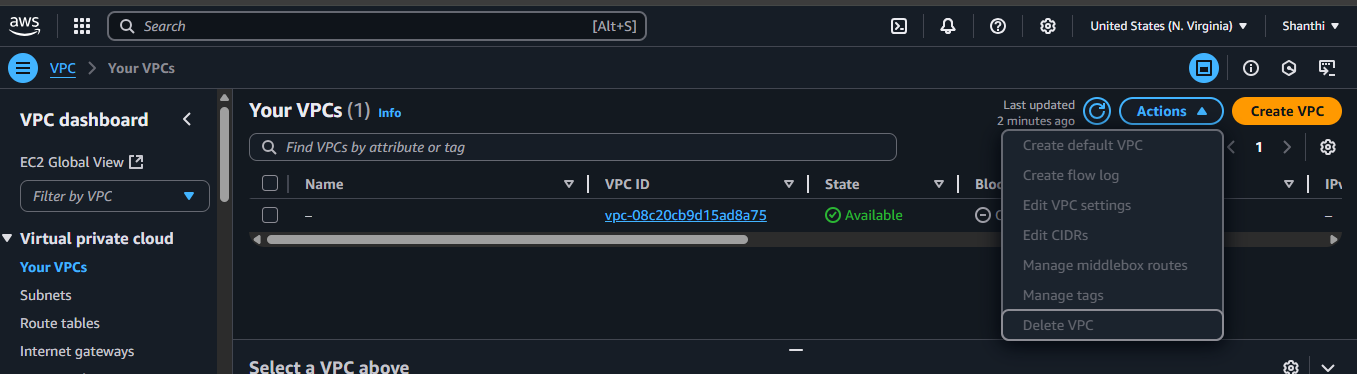
terraform apply -auto-approve



Got the error while aplying



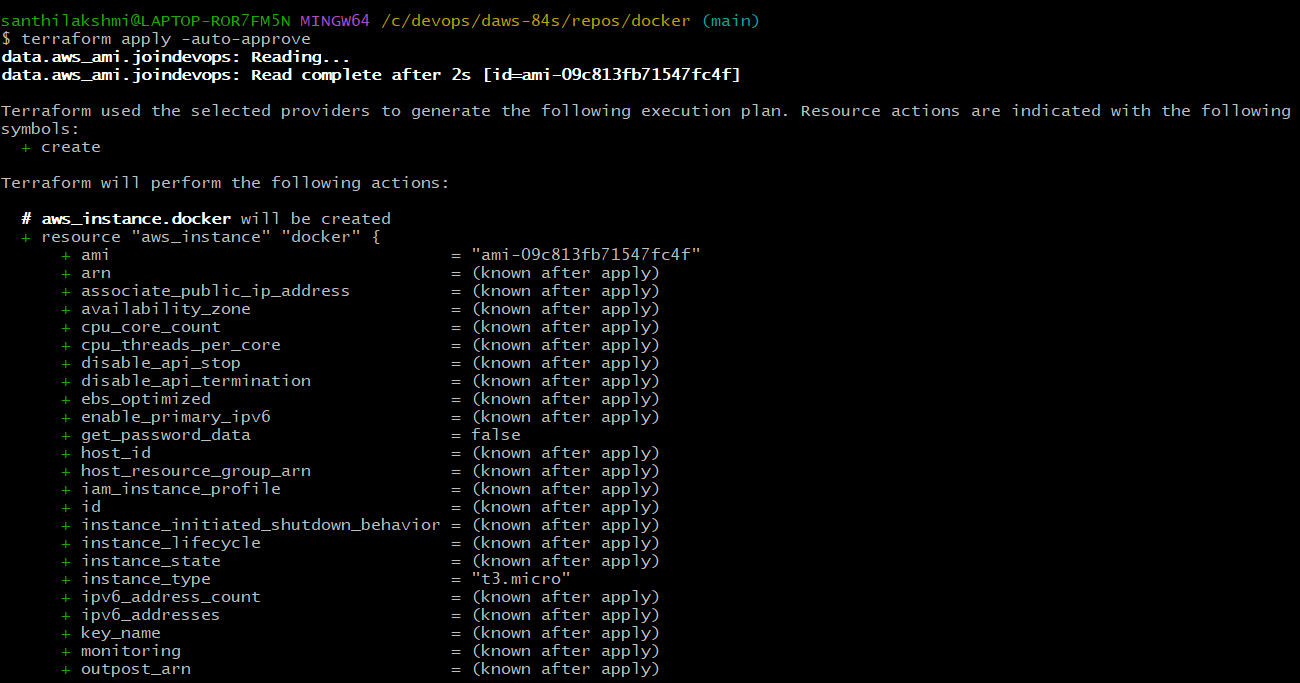
Default VPC not there in my AWS Account that’s why I got this error.

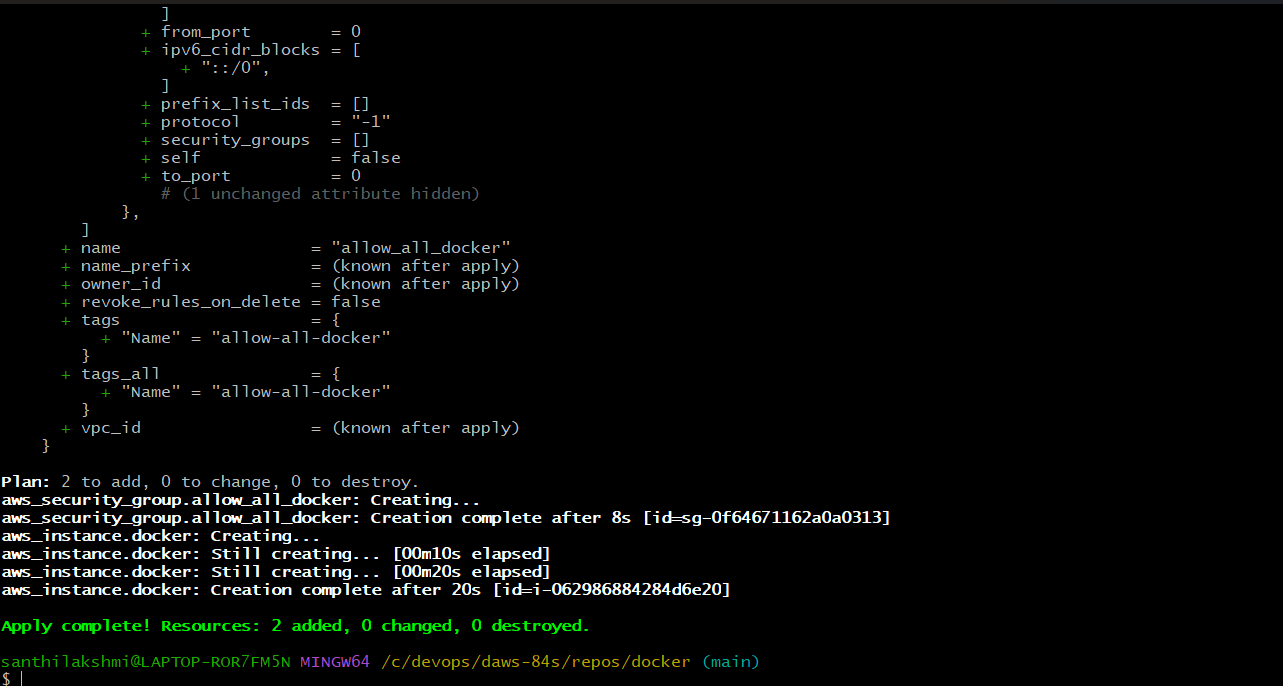


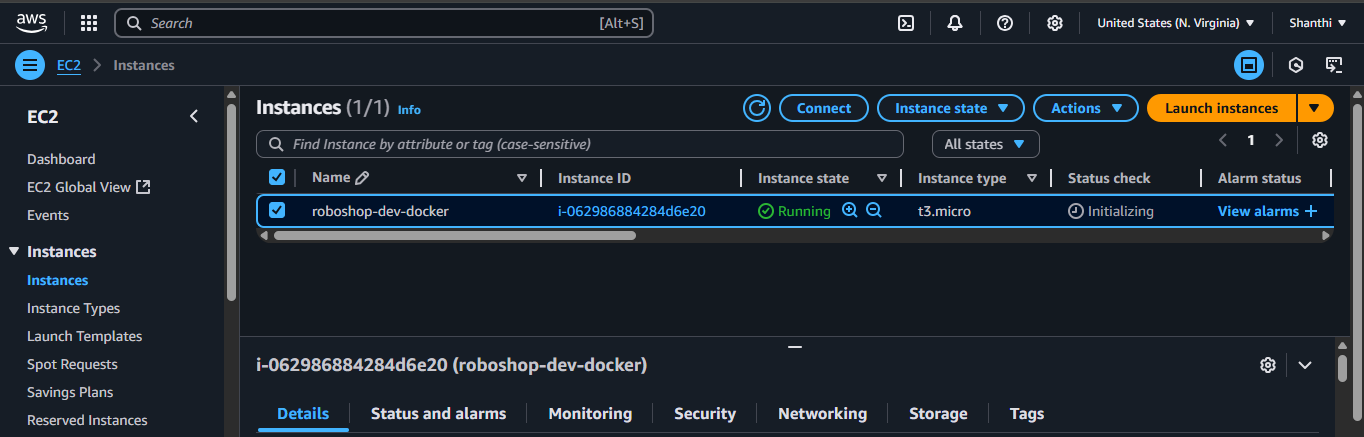
In Actions we have defailt VPC

**cd /c/devops/daws-84s/repos/docker**

**terraform apply -auto-approve**

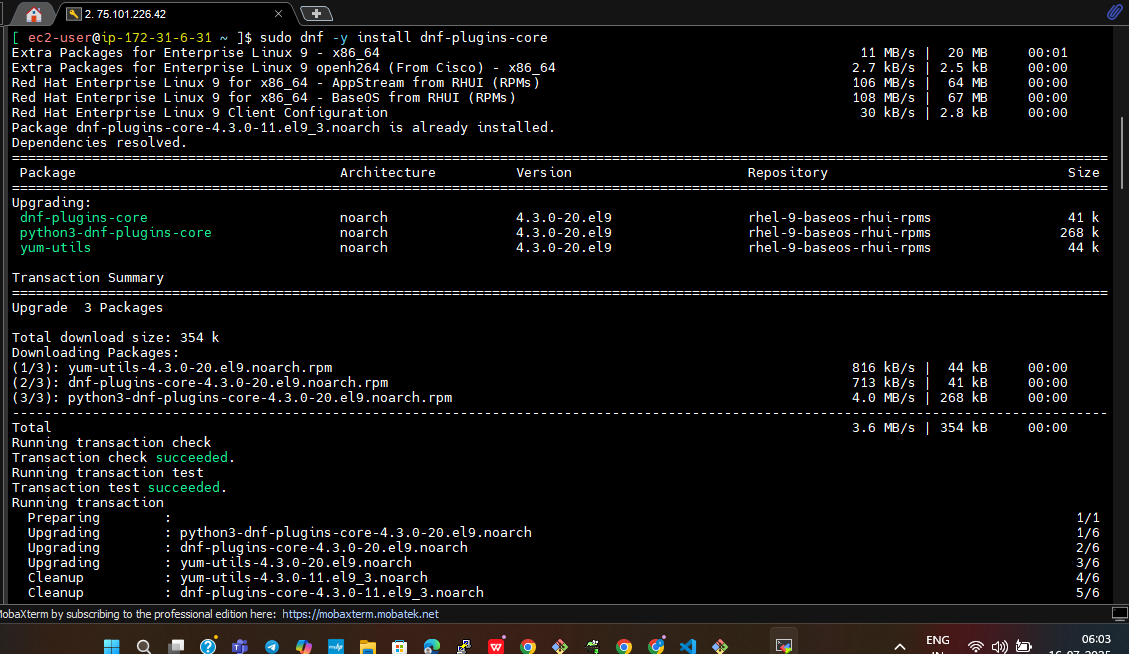






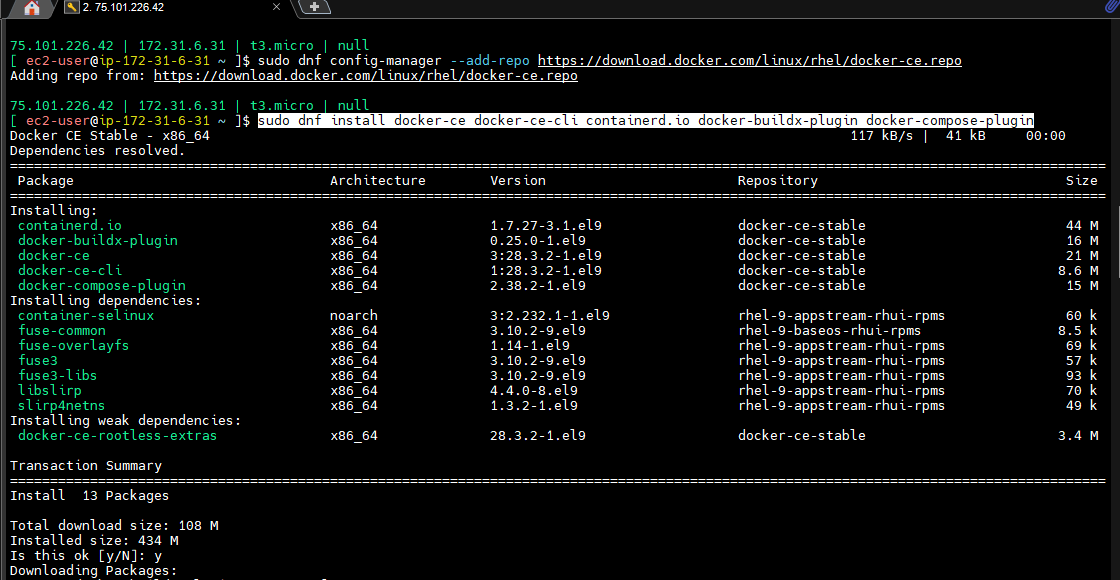
Connect the server

**sudo dnf -y install dnf-plugins-core**



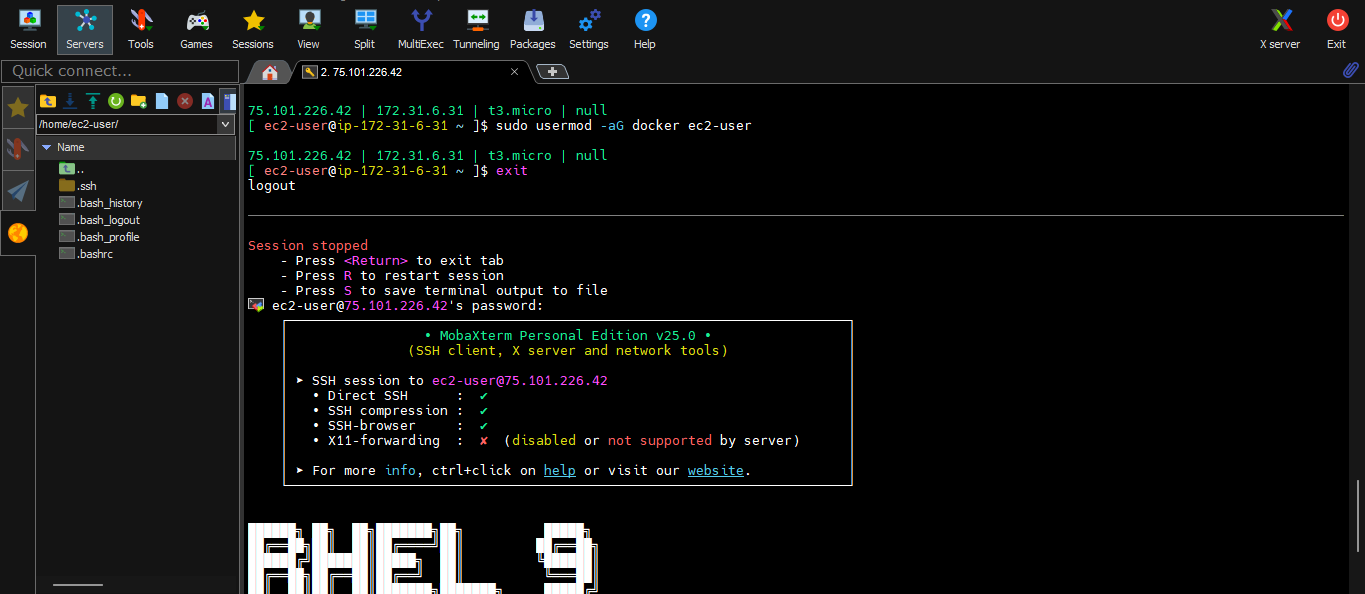
**sudo dnf config-manager --add-repo <https://download.docker.com/linux/rhel/docker-ce.repo>**

**sudo dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin**



Change the group user

**sudo usermod -aG docker ec2-user**

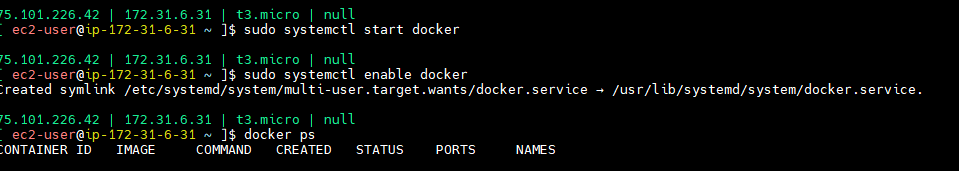


Disconnect the server and relogin the server.

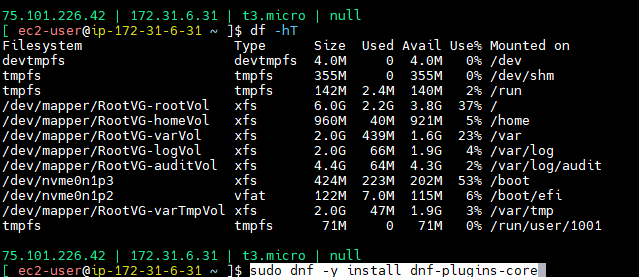
**sudo systemctl start docker**

**sudo systemctl enable docker**

**docker ps**



**df -hT**



Let’s discuss the other instances.

This is different directory dockerfile

**LABEL**

Lebel just like tag our AWS. Tag menas jey value pair, label also key value pair.

One key value pair you assign that will use in filtration. This don’t have functionality purpose just like documentation purpose. And you can use labels for at time of filtration.

Should be there base value.

**LABEL/Dockerfile**

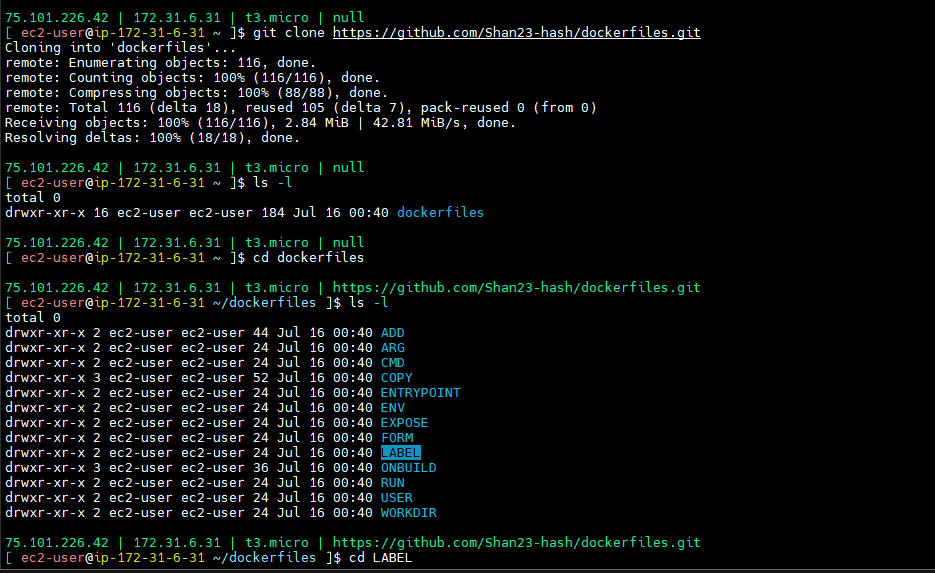
FROM almalinux:9

LABEL COURSE="DevOps" \

TRAINER="Sivakumar" \

DURATION="120HRS" \

**git clone <https://github.com/Shan23-hash/dockerfiles.git>**

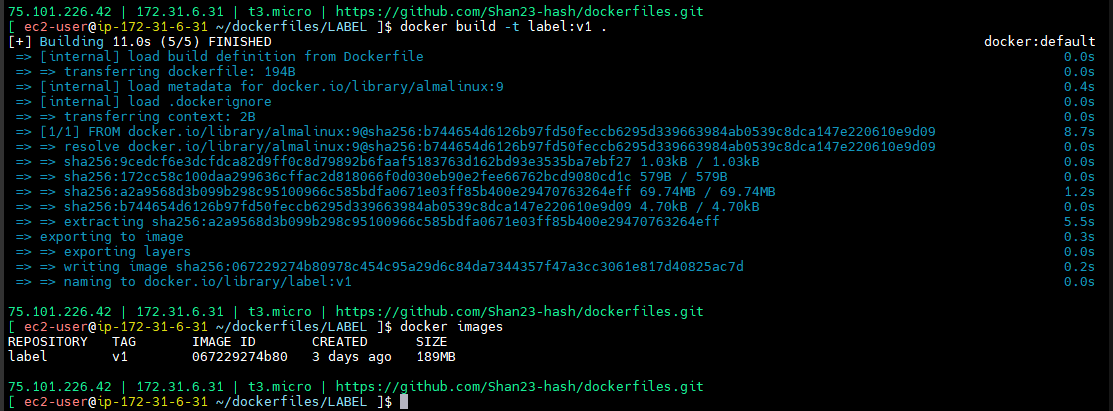


**cd LABEL**

**cd /dockerfiles/LABEL**

**docker build -t label:v1 .**

**docker images**

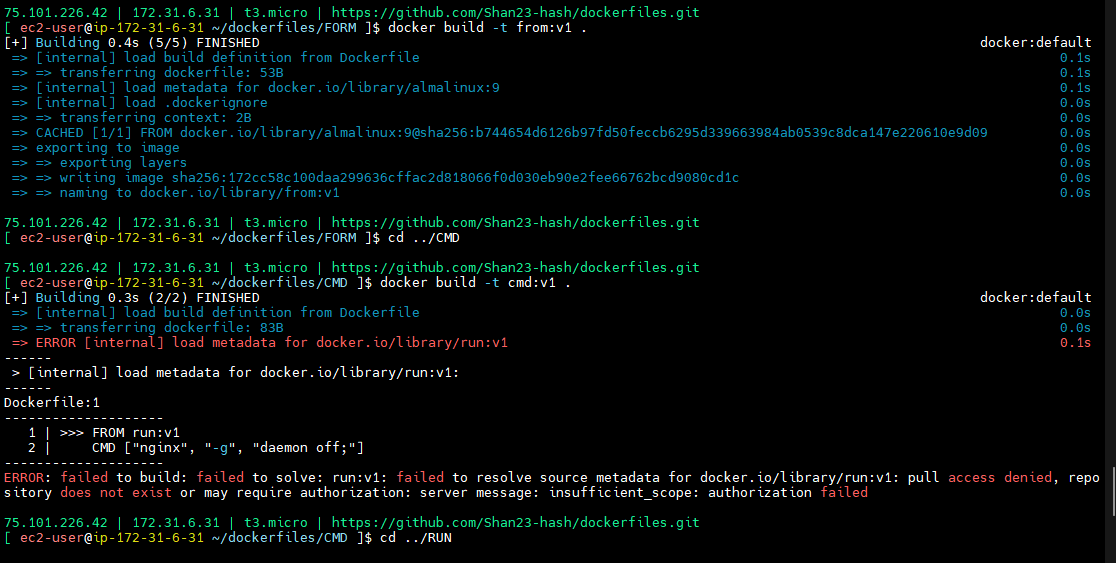


**cd /dockerfiles/FORM**

**docker build -t from:v1 .**

**cd /dockerfiles/CMD**

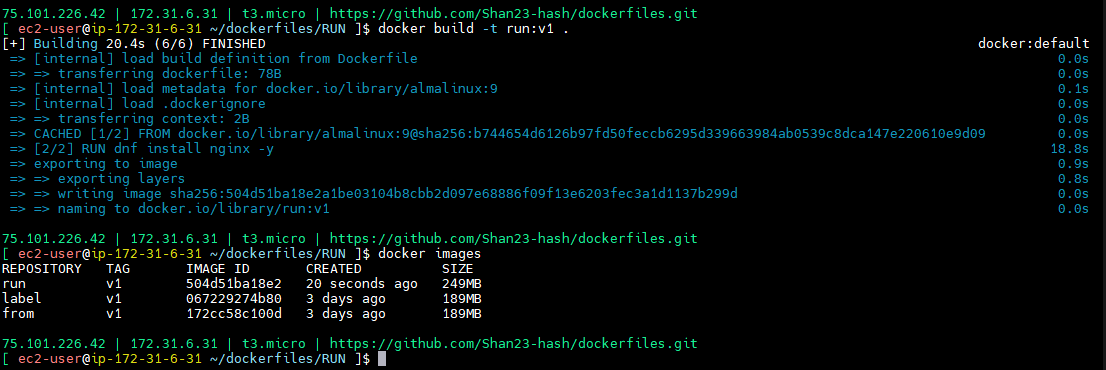
**docker build -t cmd:v1 . -- -- here got error**



**cd /dockerfiles/RUN**

**docker build -t run:v1 .**

**docker images**



You just imagine 100 images are there. How can you filter the images based on labels

**docker images --filter "label=COURSE=DevOps"**

Same like this will use in filters.

Key value pair will add one meta data.

Maintainer -- deprecate the maintainer wear into labels who is author here like that mentioned here.

**LABEL/Dockerfile**

FROM almalinux:9

LABEL COURSE="DevOps" \

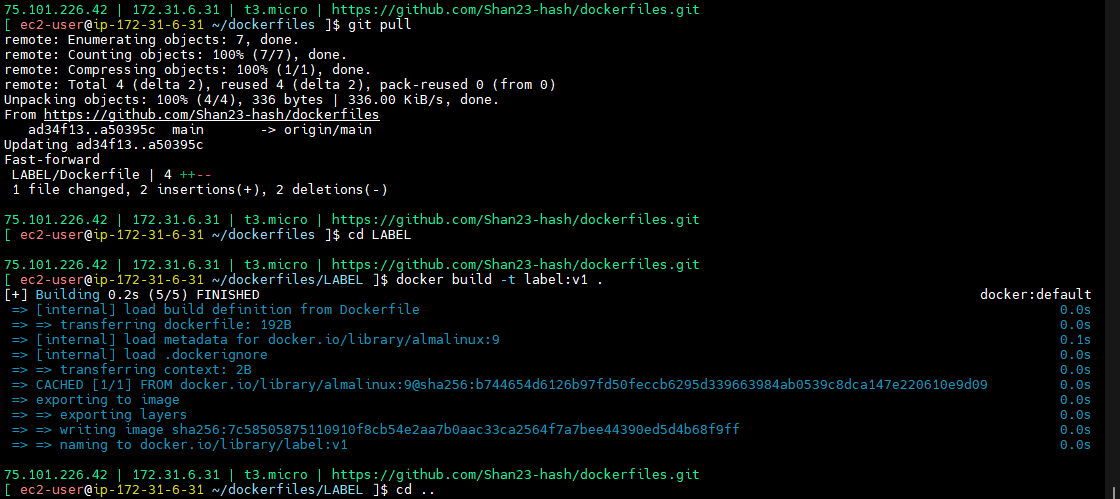
TRAINER="Sivakumar" \

DURATION="120HRS" \

[com.joindevops.dockerfile.author="shanbojja@gmail.com"](mailto:com.joindevops.dockerfile.author=%22siva@joindevops.com%22)

**cd ~/dockerfiles/LABEL**

**docker build -t label:v1 .**



How can push this images into dockerhub

First we have login in dockerfile.

docker push URL /username/image-name:version

Private repository also we can create. In some other place.

Dockerhub will control in docker.

We can’t keep all the images in dockerhub. Have security issue.

So that’s why in companies we have private ip in nexes,ecr

In hub

docker push docker.io by default it will take

If incase you are using any other URL you have to give That URL

docker push docker.io/shan2324/label:v1

Docker repository - nexus,openshift,harbar like this there more repos

If any one procedure same

If you are pushing docker no need to give to dockerhub

Otherwise you should give user name

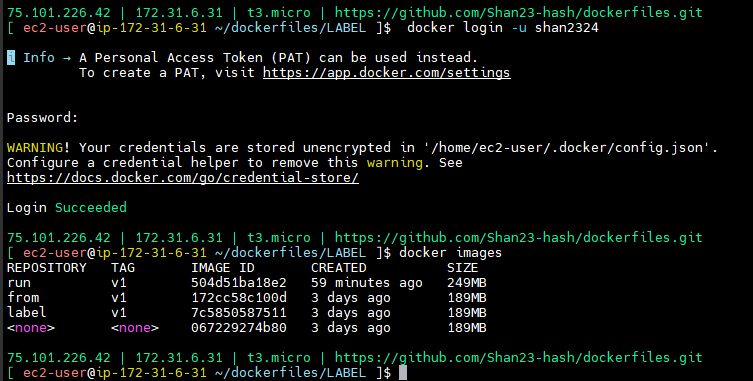
docker push docker.io/shan2324/label:v1

If you are not given user name they can’t understand where have push

If you want push login first

**docker login -u shan2324** (logged in successfully)

**docker images**



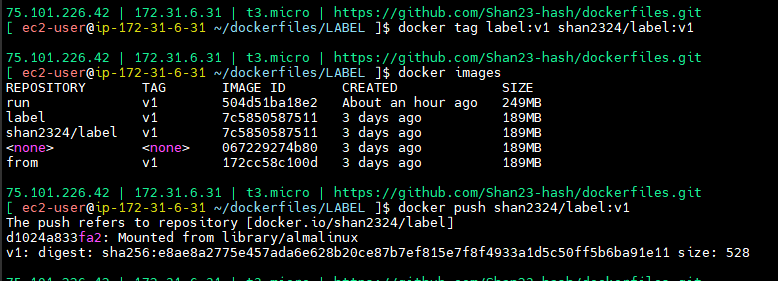
If you given docker push label -- it will not push

Retag first image

**docker tag label:v1 shan2324/label:v1**

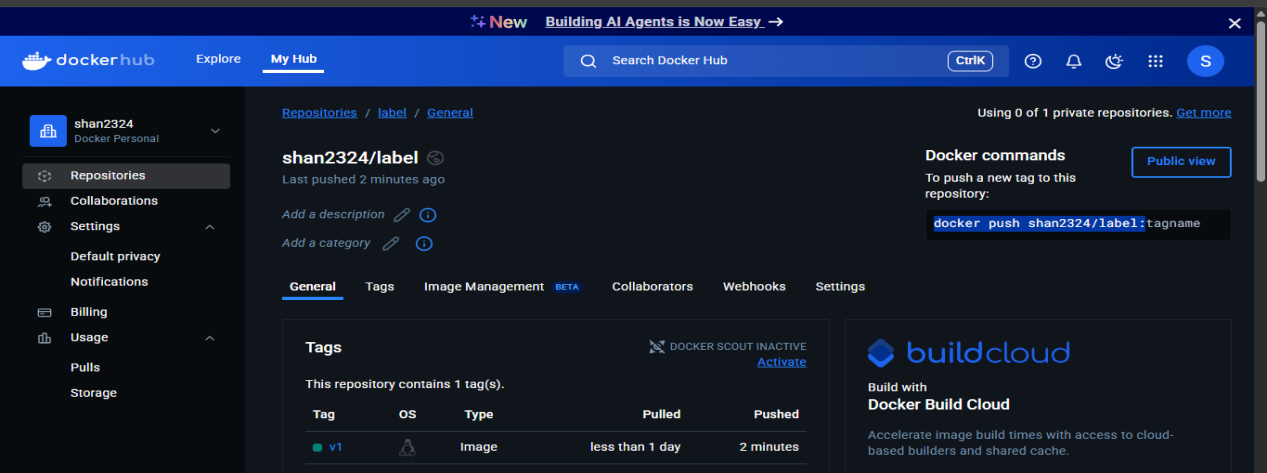
**docker images**

**docker push shan2324/label:v1**



Go to dockerhub

Check my hub



**EXPOSE**

Expose instruction functionality not added for documentation purpose.

Your image which port it’s opening. Atleast inform to users you can use expose

Nginx,cart,catalogue - all are customized applications.this all which port will open don’t no anyone generally. We will use our wish port but that have to know public means have to use expose.

That is purely documentation purpose not any functionality.

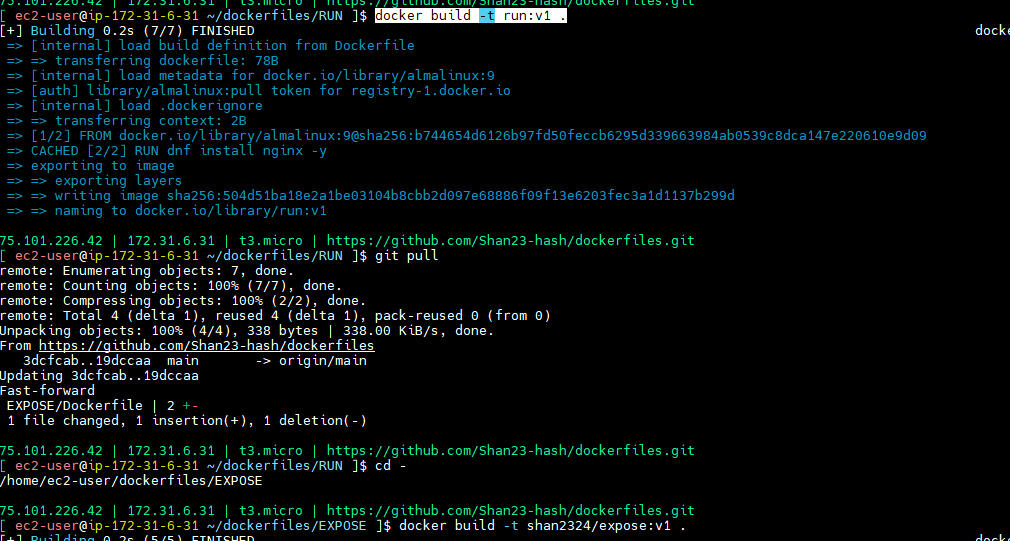
FROM run:v1

EXPOSE 80

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

**cd ~/dockerfiles/RUN**

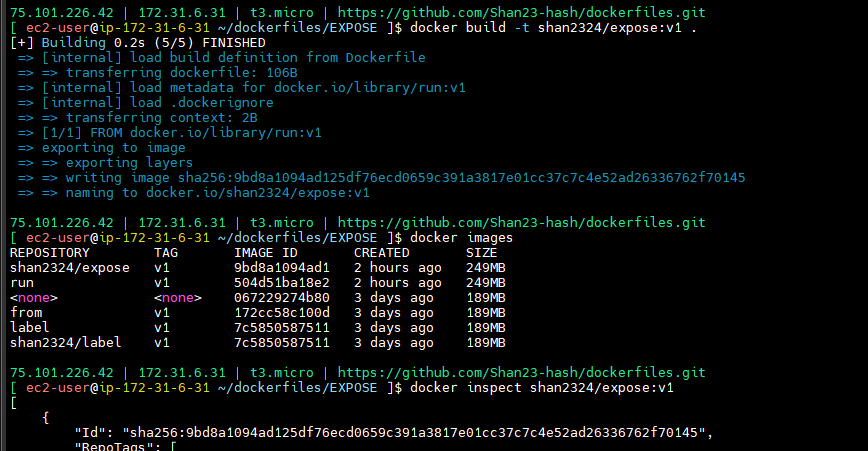
**docker build -t run:v1 .**



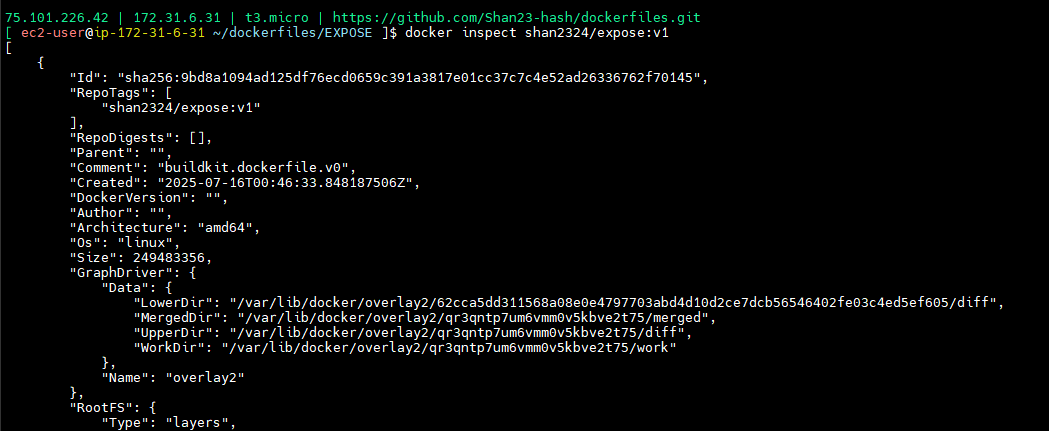
**cd ~/dockerfiles/EXPOSE**

**docker build -t shan2324/expose:v1 .**

**docker images**



**docker inspect shan2324/expose:v1**

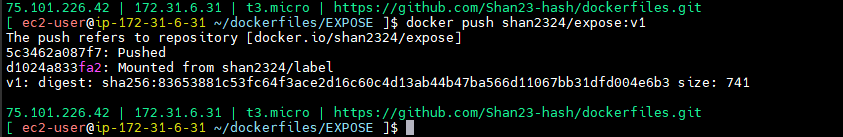


Here we have information

Purely of documentation . they don’t kmnow which port it’s opening.

Let’s up to us.

Command for pushing - **docker push shan2324/expose:v1**



FROM shan232/run:v1

#FROM run:v1

EXPOSE 80

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

If you given like this - if incase not there in local it will take from github

**ENV**

Environment variables we know same like the linux variables. You can access inside the server.

Ex:Java program running In linux so it can access linux variable environment and use them.

ENV/Dockerfile

FROM almalinux:9

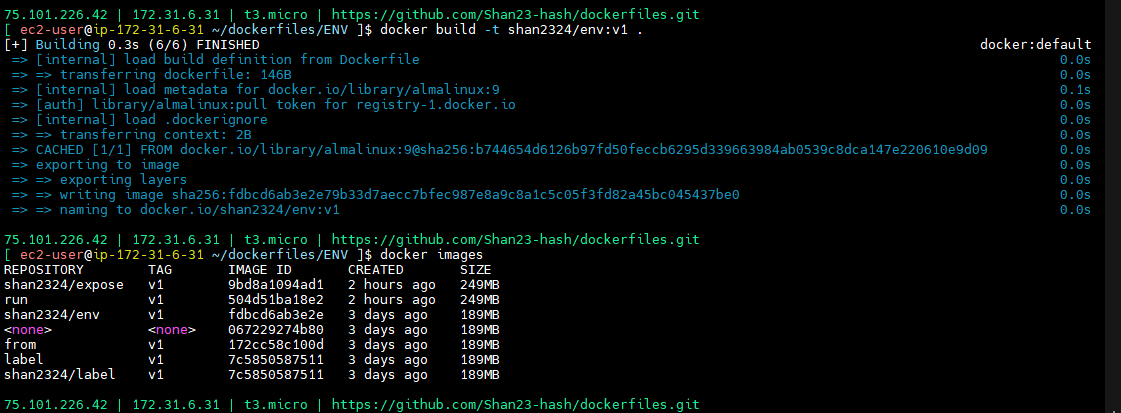
ENV COURSE="DevOps" \

TRAINER="Sivakumar" \

DURATION="120HRS"

**docker build -t shan2324/env:v1 .**

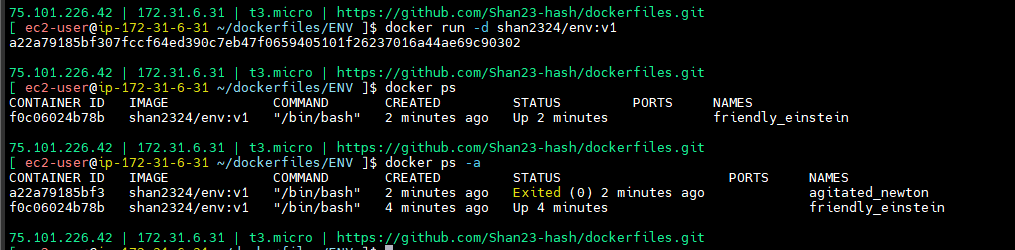
**docker images**



**docker run -d shan2324/env:v1**

**docker ps**

**docker ps -a**



Why that was exited we are not given any command for running

One is running nginx what is that status -- continuous running

Continuous running Browser is - there is nothing

FROM almalinux:9

ENV COURSE="DevOps" \

TRAINER="Sivakumar" \

DURATION="120HRS"

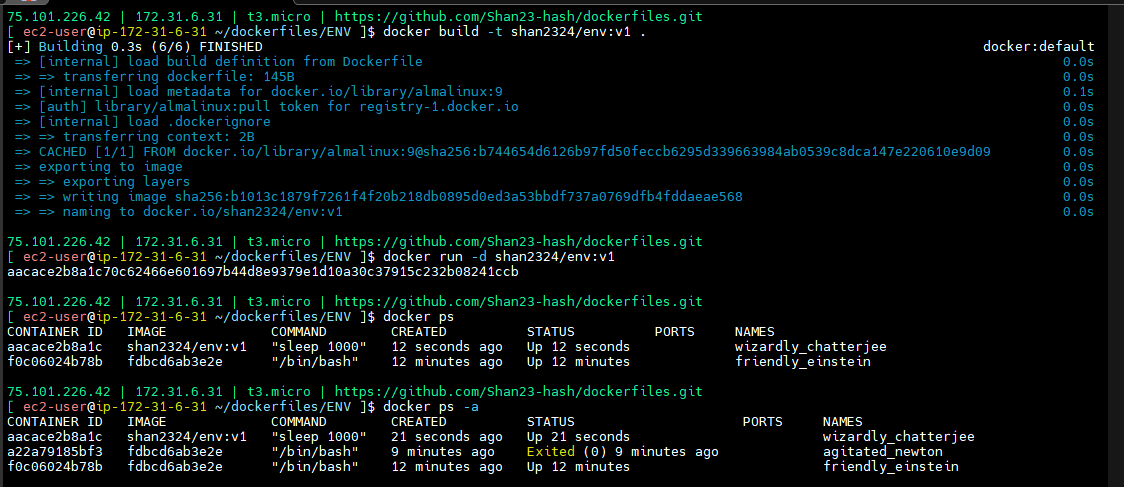
CMD ["sleep", "1000"]

**docker build -t shan2324/env:v1 .**

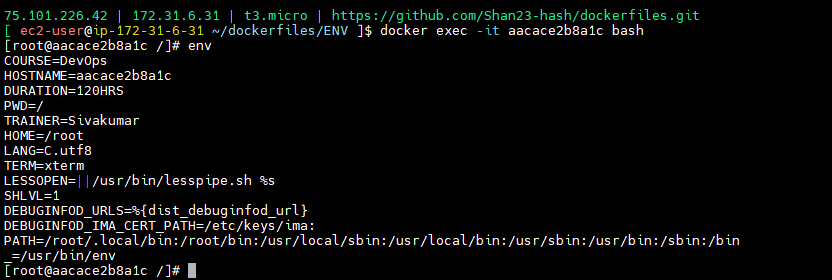
**docker run -d shan2324/env:v1**

**docker ps**

**docker ps -a**



**docker exec -it aacace2b8a1c bash**



Like this we set the environment variables inside running code they can access there variables.

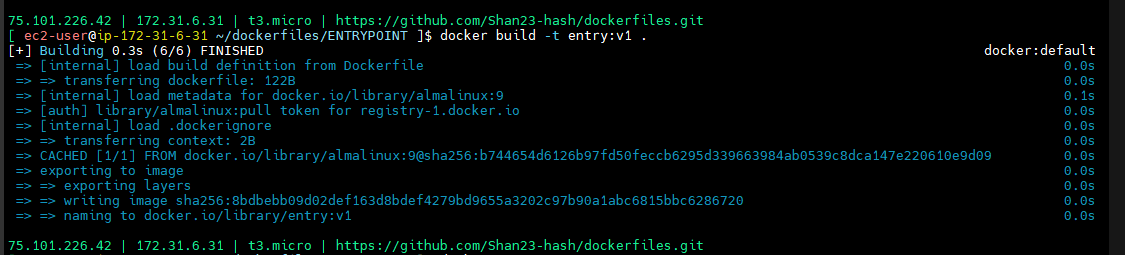
**ENTRYPOINT**

**ENTRYPOINT/Dockerfile**

FROM almalinux:9

CMD ["ping","google.com"]

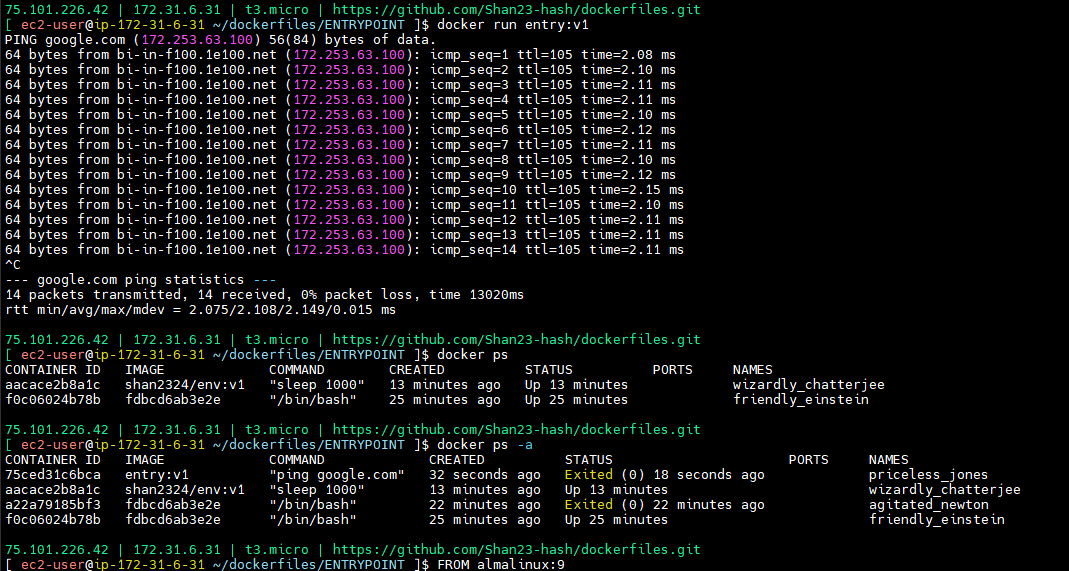
**docker build -t entry:v1 .**



**docker run entry:v1**

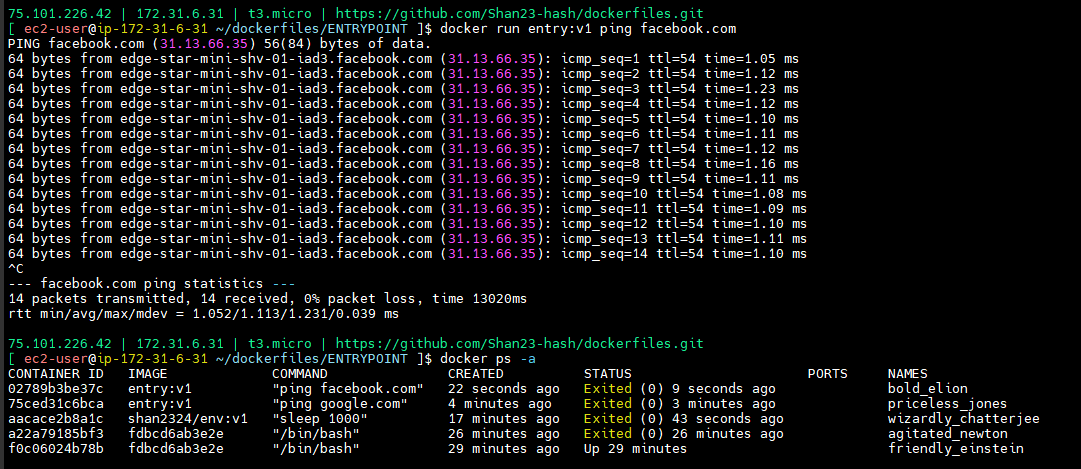
**docker ps**

**docker ps -a**



**docker run entry:v1 ping facebook.com**

**docker ps -a**



**Can I override cmd instruction in entrypoint? (imp)**

* CMD instruction can be overridden
* We can’t override ENTRYPOINT, if we try to do it will override.
* We can use combination of CMD and ENTRYPOINT for better results, ENTRYPOINT will have command, default arguments or options can be supplied by CMD.
* You can always override default args thriugh command line.

Because we can override CMD command.

Same thing we can change entry and will see

**ENTRYPOINT/Dockerfile**

FROM almalinux:9

#CMD ["ping","google.com"]

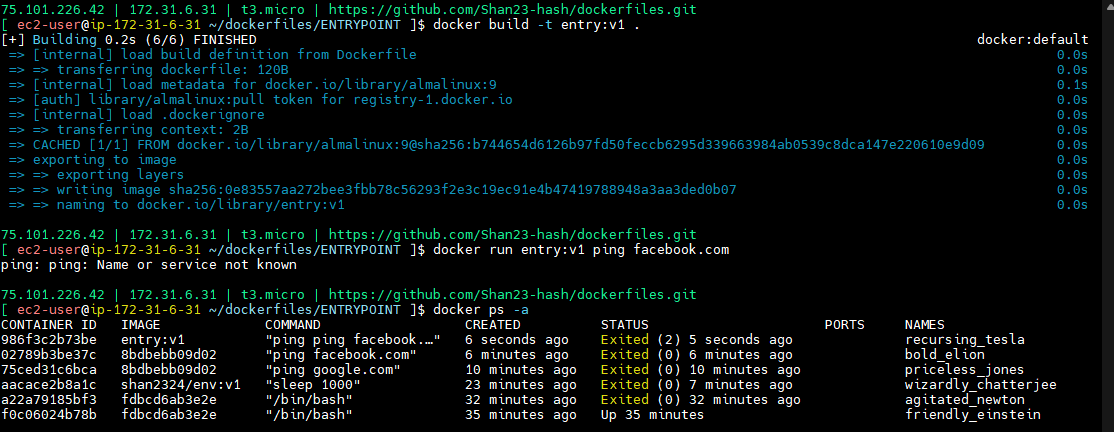
CMD ["google.com"]

ENTRYPOINT ["ping"]

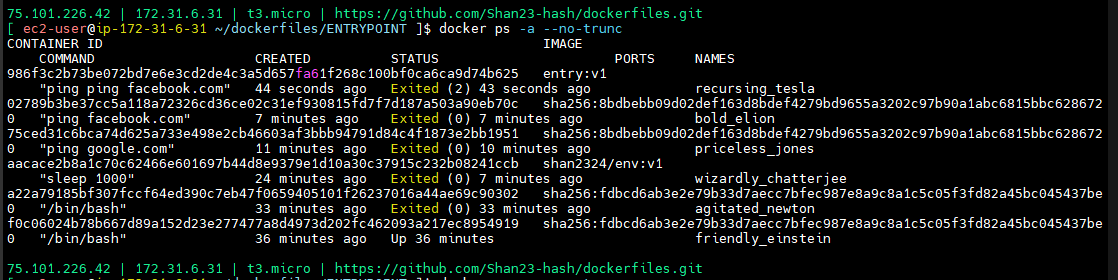
**docker build -t entry:v1 .**

**docker run entry:v1 ping facebook.com**

**docker ps -a**



**docker ps -a --no-trunc**

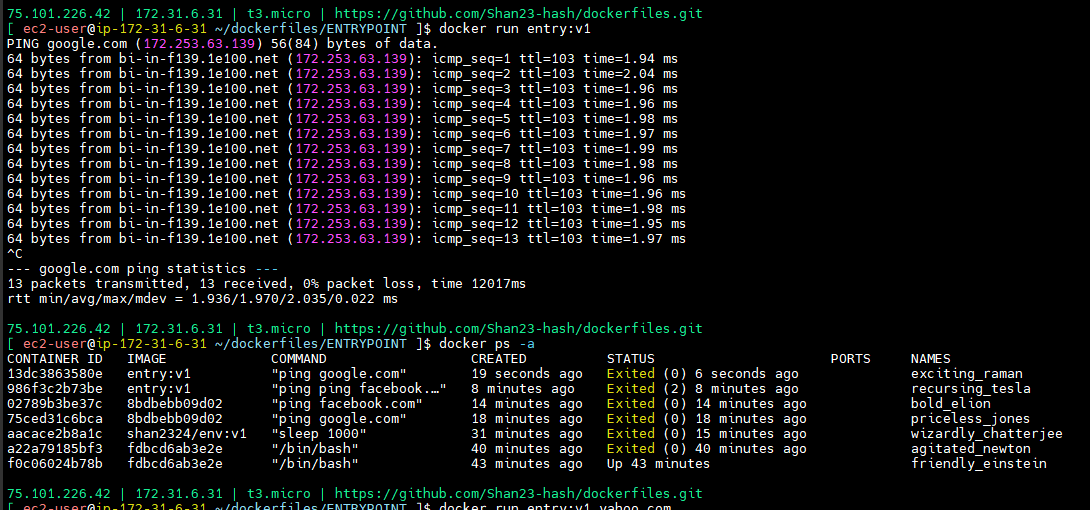


We can’t overide, appending only happeing.

If you are not given it’s taking google.com

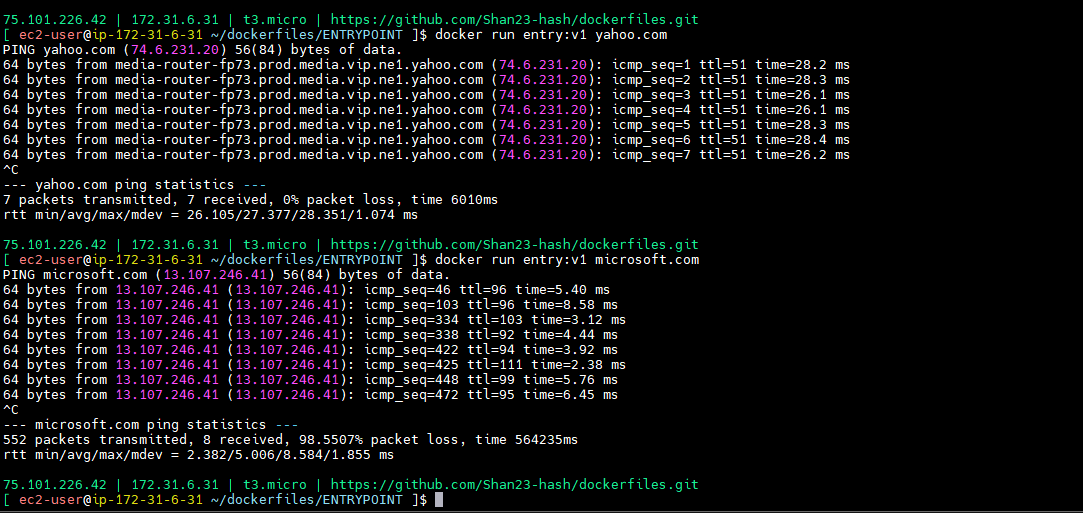
**docker run entry:v1**

**docker ps -a**



Otherwise you can give yahoo.com

**docker run entry:v1 yahoo.com**



Security also there - I can’t override the command

I can only supply the options and arguments through the command prompt.

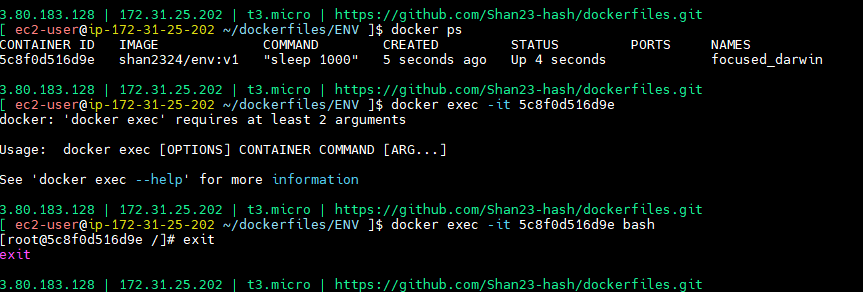
ENTRYPOINT also same like CMD. One more option they given.

**USER**

Here **docker ps**

Any running inside while login what is the access I’m getting

**docker exec -it 5c8f0d516d9e bash**

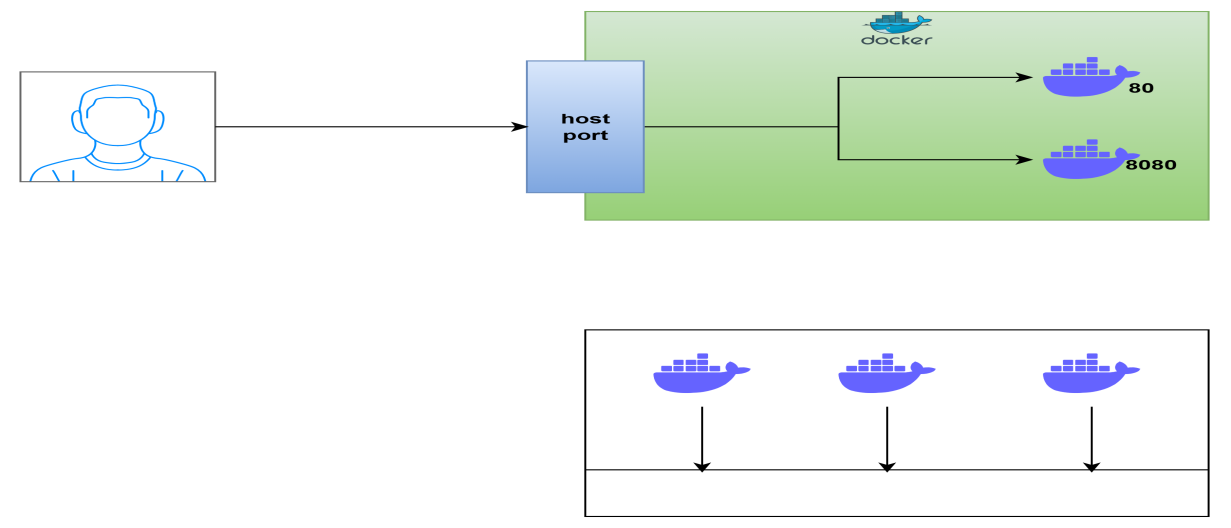


Root access - if root access there that is a problem

How it is problem

**Diagram**

**-------------**



This is the storage.

Containers was running.

Containers have any saparate storage? Are else using Host server only?

In Which thing it’s running same storage will take.

I given root access to one container-- it can belong to entire storage.

We given root access it may do anything.

**USER/Dockerfile**

FROM almalinux:9

#CMD ["ping","google.com"]

RUN dnf install git -y

RUN useradd sivakumar

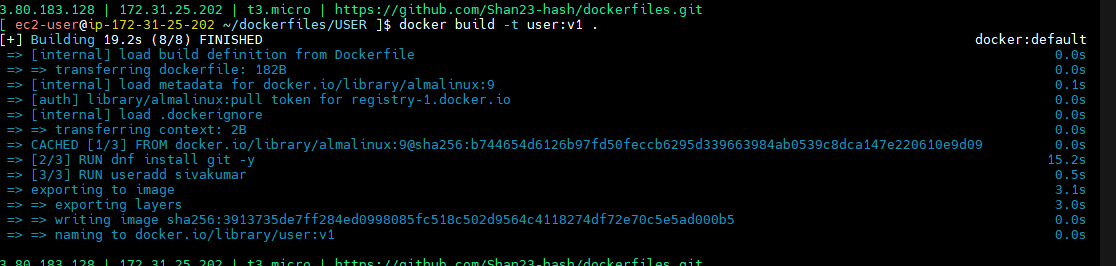
USER sivakumar

CMD ["google.com"]

ENTRYPOINT ["ping"]

**cd ~/dockerfiles/USER**

**docker build -t user:v1 .**

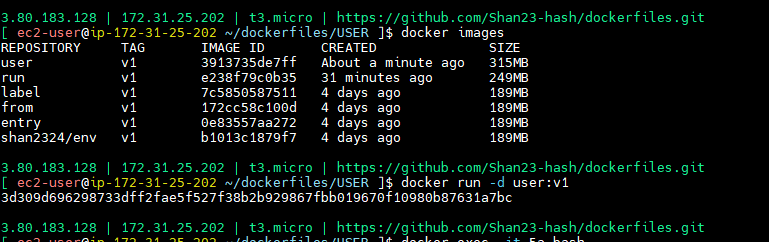


Whatever you want install it and configure it.because you have root access install those. But finally give useradd sivakumar.

**docker images**

Now we have docker images that user

**docker run -d user:v1**



**docker ps**

**docker exec -it 9cd98dab42ea bash**



Now install anything

**Imp::::**

While building docker Images, while doing our roboshop project will changing to docker images and while creating this is very important without using root user you can install normal user.

--> ADD and CMD it will add code from local to container.

--> CMD it will the container running

--> Entrypoint also same and it has the difference same devices entrypoint

You can mix both for better results.

--> ENV key value pairs inside we can aceess it.

--> EXPOSE - just port information only for documentation purpose.

--> FROM should be the first instruction.

--> LABEL it will give key value pairs use for filteration purpose.

--> RUN use to configure the image like installing packages etc.

--> USER it will keep the container as normal user. Which user we running containers.

--> WORKDIR which directory we are in.

**WORKDIR**

**WORKDIR/Dockerfile**

FROM almalinux:9

RUN mkdir /tmp/workdir-demo

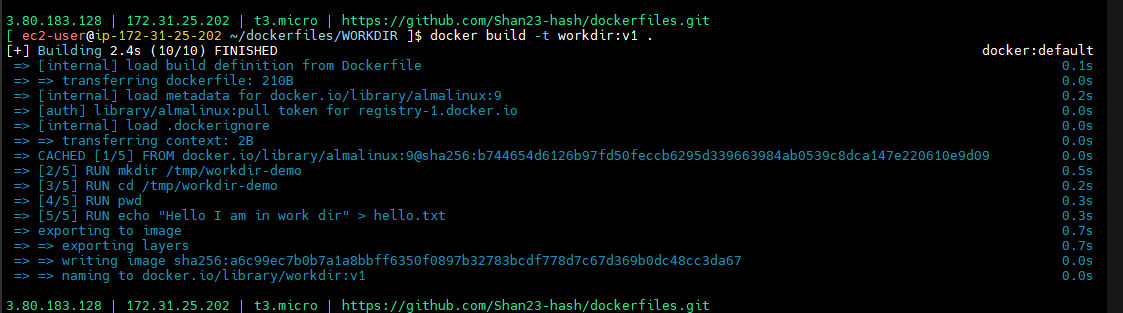
RUN cd /tmp/workdir-demo

RUN pwd

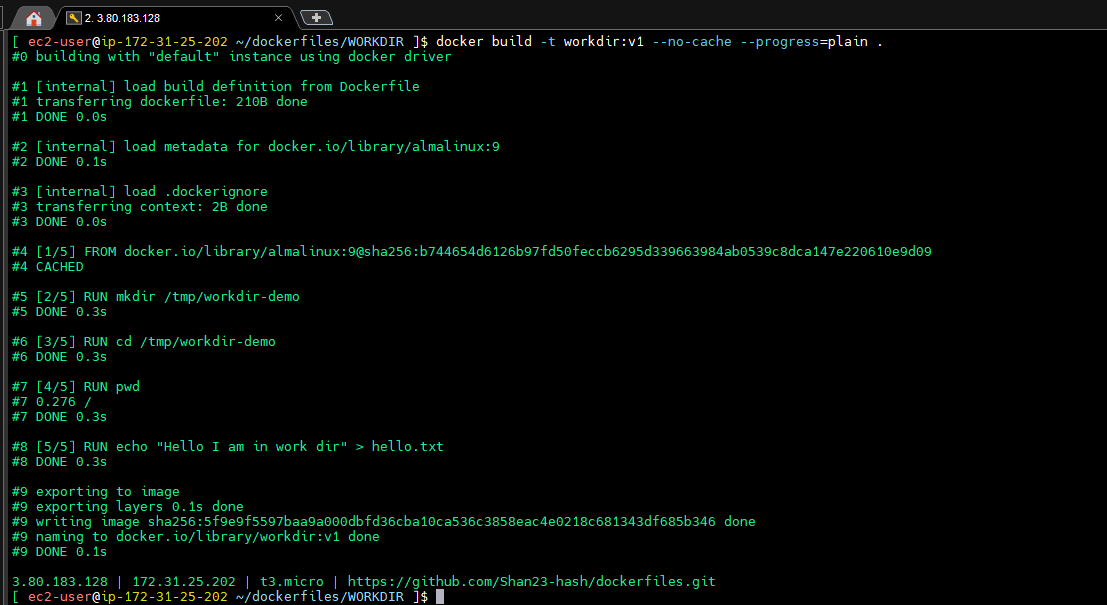
RUN echo "Hello I am in work dir" > hello.txt

CMD ["sleep","1000"]

**docker build -t workdir:v1 .**



**docker build -t workdir:v1 --no-cache --progress=plain .**



It will not work our cd command so here we will use workdir

**WORKDIR/Dockerfile**

FROM almalinux:9

RUN mkdir /tmp/workdir-demo

#RUN cd /tmp/workdir-demo

WORKDIR /tmp/workdir-demo

RUN pwd

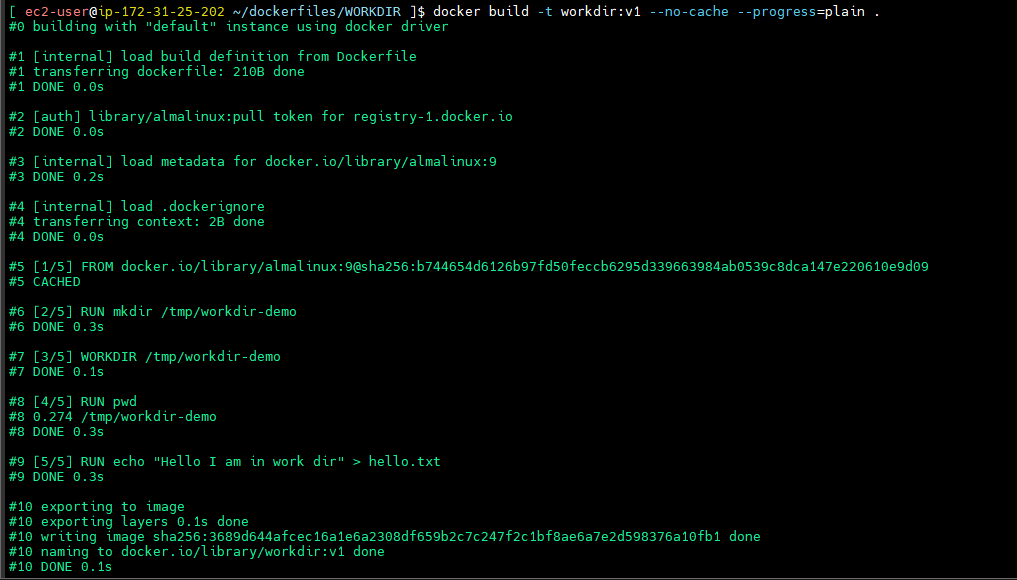
RUN echo "Hello I am in work dir" > hello.txt

CMD ["sleep","1000"]

It we given workdir setuping one directory everything will do here only.

Set uping your working directory.

**docker build -t workdir:v1 --no-cache --progress=plain .**

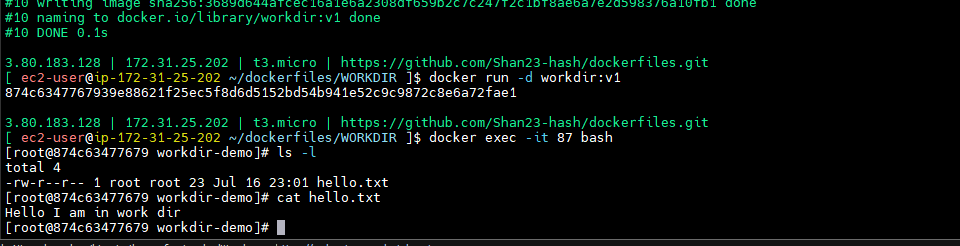


**docker run -d workdir:v1**

**docker exec -it 87 bash**

**ls -l**

**cat hello.txt**



**ARG**

ARG also just like same env key value pairs but one small difference is there.

**ARG/Dockerfile**

FROM almalinux:9

ENV course="devops" \

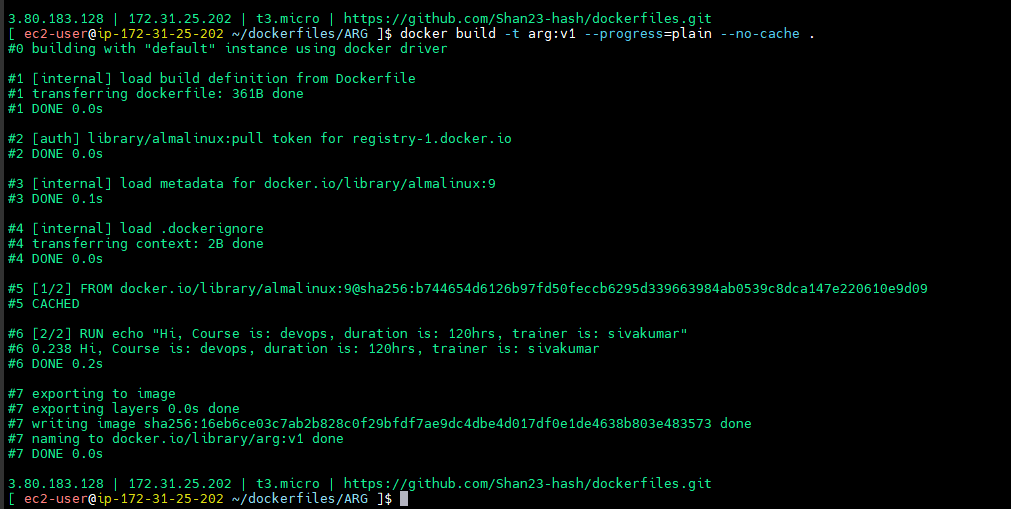
    duration="120hrs"

ARG trainer="sivakumar"

RUN echo "Hi, Course is: ${course}, duration is: ${duration}, trainer is: ${trainer}"

CMD ["sleep", "1000"]

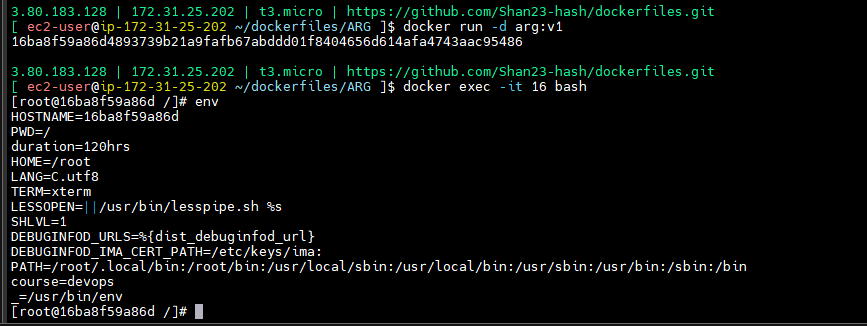
**docker build -t arg:v1 --progress=plain --no-cache .**



**docker run -d arg:v1**

**docker exec -it 16 bash**

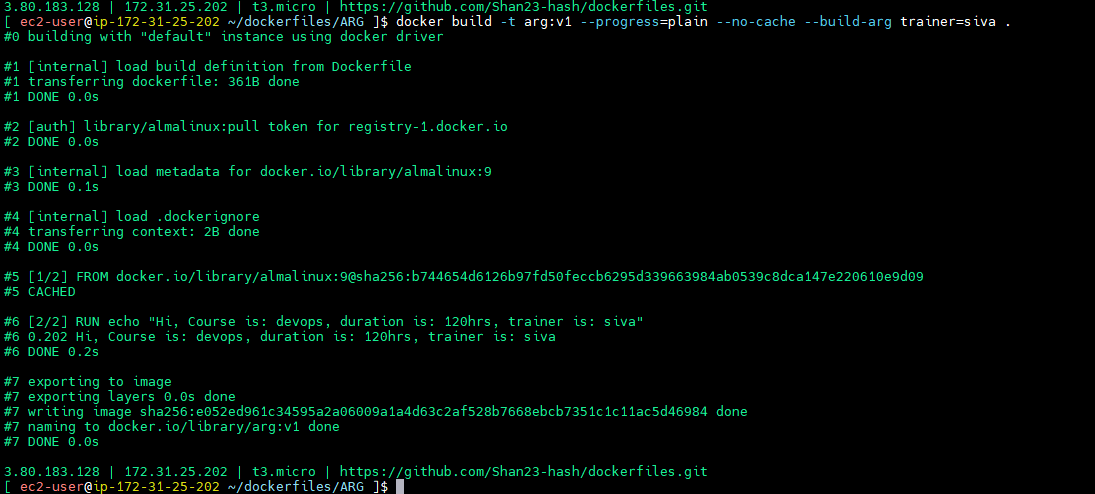
**env**



**ARG is build time variables, they can’t be accessed inside container. ENV can be accessed build time and inside container also.**

I can override it also

**docker build -t arg:v1 --progress=plain --no-cache --build-arg trainer=siva .**



ARG instruction variables can be overridden.

What should be the first instruction in docker file -- FORM but there is exception arg we can be the first instruction to supply the version

**ARG/Dockerfile**

ARG version

FROM almalinux:${version:-9}

ENV course="devops" \

duration="120hrs"

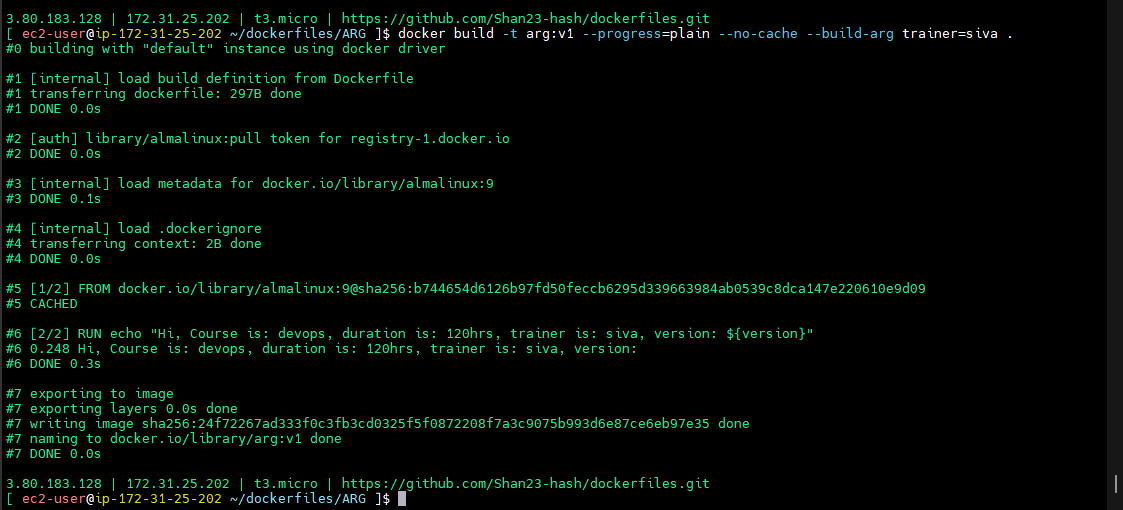
ARG trainer="sivakumar"

env

RUN echo "Hi, Course is: ${course}, duration is: ${duration}, trainer is: ${trainer}, version: ${version}"

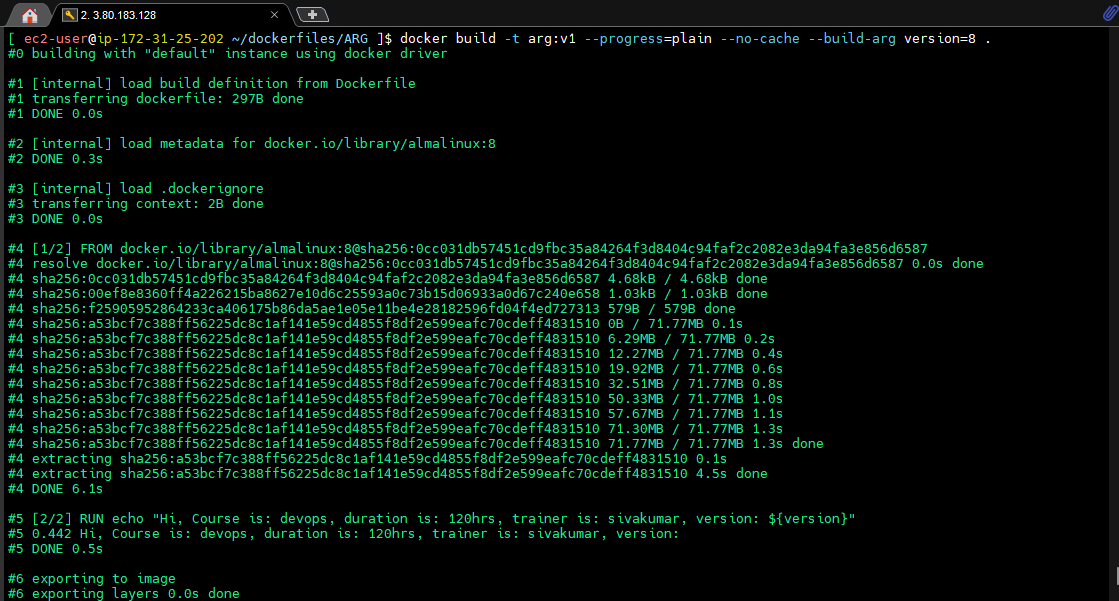
CMD ["sleep", "1000"]

**docker build -t arg:v1 --progress=plain --no-cache --build-arg trainer=siva .**



I can give 8

**docker build -t arg:v1 --progress=plain --no-cache --build-arg version=8 .**



Future it will come 10 also no need to code you can supply here also

--> Usually FROM should be the first instruction but in exceptional case ARG can be the first instruction to provide the version to the base OS.

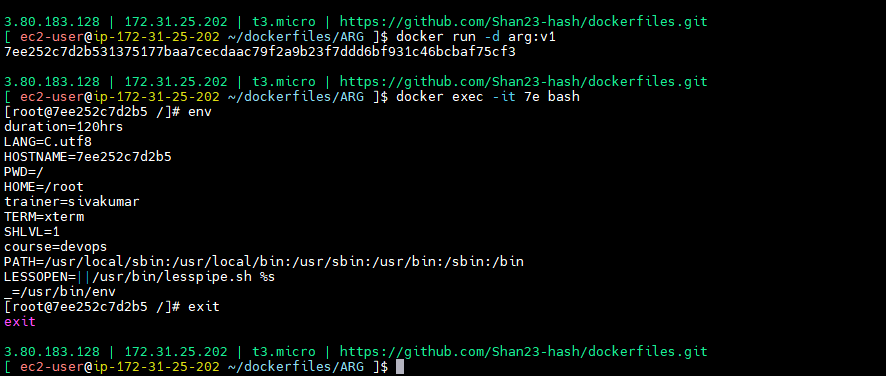
--> in an exceptional case ARG can be the first instruction to supply version to base OS in FROM, you cant use that version after FROM instruction

How can you use ARG inside container?(int que)

**docker run -d arg:v1**

**docker exec -it 7e bash**

**Env**



**ONBUILD**

I created image it can use some other peoples

While devloping images you can put some conditions while others are using your images…

I’m building one nginx in my organization.

Container id no need to give fully give upto unique

**ONBUILD/Dockerfile**

FROM almalinux:9

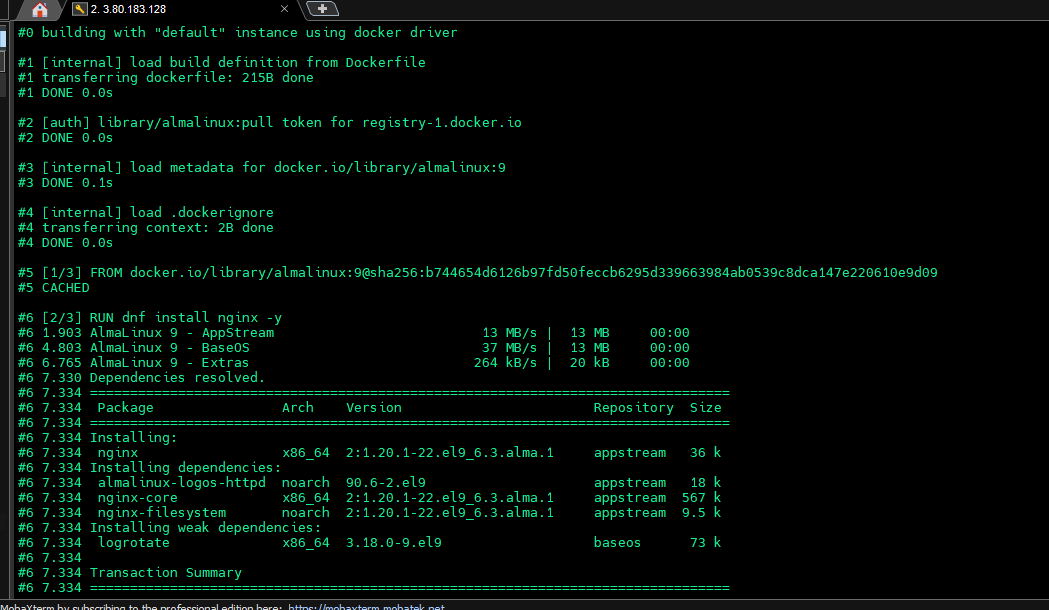
RUN dnf install nginx -y

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

ONBUILD COPY index.html /usr/share/nginx/html/index.html

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

**docker build -t onbuild:v1 --progress=plain --no-cache .**



Onbuild instruction not run

Id someone used my used then it will run

**ONBUILD/test/Dockerfile**

FROM onbuild:v1

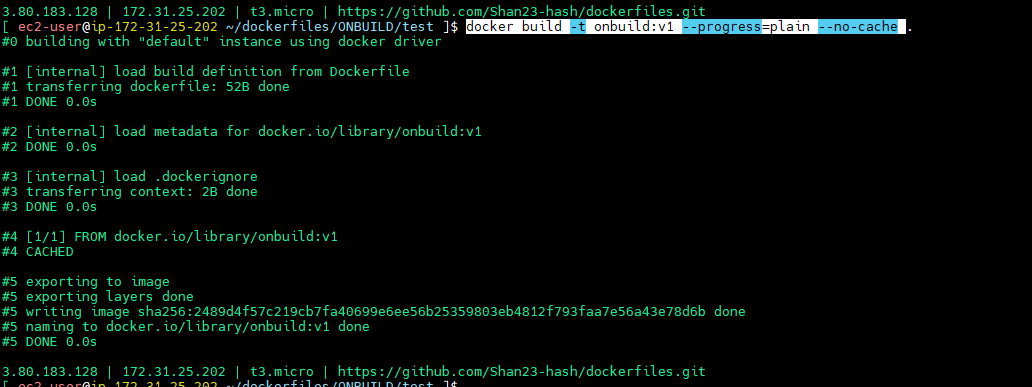
I’m forcing the users of my village they should have index.html

**ONBUILD/test/index.html**

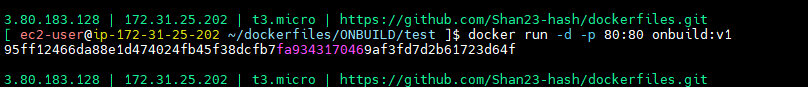
<h1>Hi this is onbuild</h1>

**cd ~/dockerfiles/ONBUILD/test**

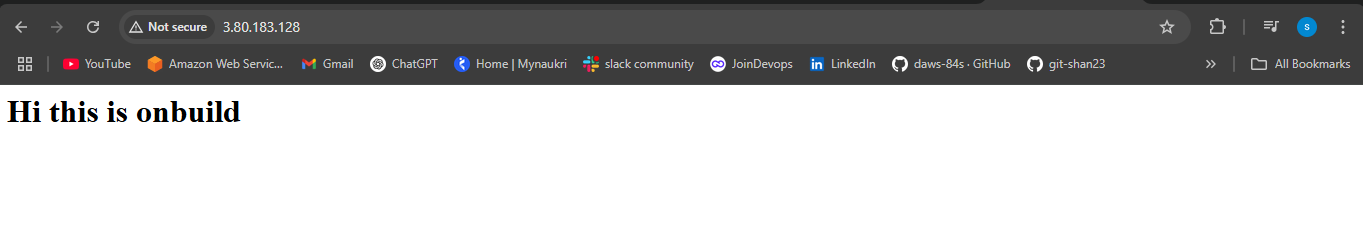
**docker build -t onbuild:v1 --progress=plain --no-cache .**



**docker run -d -p 80:80 onbuild:v1**



<http://3.80.183.128/>



The ONBUILD instruction in a Dockerfile adds a trigger instruction to an image, which will be executed at a later time when that image is used as the base for another build. This mechanism is particularly useful for creating reusable base images that automatically perform common setup tasks for derivative images.

How ONBUILD works:

**Defining ONBUILD in a base image:**

When you create a Dockerfile for a base image, you can include ONBUILD instructions. For example, an ONBUILD instruction might copy application files, install dependencies, or set environment variables.

**Building a derivative image:**

When you then use this base image with FROM in a new Dockerfile, the ONBUILD instructions from the base image are automatically executed before any instructions in the new Dockerfile.

**No execution during initial build:**

The ONBUILD instructions are not executed when the base image itself is built. They are only triggered when a new image is built from that base image.

**Dockerfile Instructions**

--> **ADD** - in local files and folders to copy the in docker images.

--> **COPY** - same like add but extra advantage we can download the files from inetrnet.it can directly enter the files into the docker images.

--> **ARG** - in build time will provide images to docker image. ARG has some speciality it can be the first instruction before to supply the version to the from instruction.  
--> **CMD** -- container started time it will run. It can be overridden.

--> **ENTRYPOINT** -- it can not be overridden for better we can use cmd entrypoint together. Entry point you can mention the command cmd can provide the arguments it can always override documents or command line.

--> **ENV** - environment variables

--> EXPOSE - it will provide the information of the ports open by the container by the user.

--> **LABEL** - key value pairs

--> ONBUILD - image developer can put something conditions through the onbuilding sections.

--> **RUN** - runs at the time of image building. Can have multiple run instructions usually we use run instruction to install few packages or to configure the image.

--> **USER** - which user the container runs, can usually not root users we mentioned user instruction because will use in security purpose.

--> **WORKDIR** - this instruction we used to mentioned the working directory inside image or container.