Summary:

- Infrastructure creation.
- Pipeline creation.
- Deployment.
- Nginx and DNS Configuration.

The source code and the required files are available in the repo:

https://github.com/Murali90102/sample_node_app.git.

First of all, clone the files in the repository to the system.

Open the Terraform folder in a terminal and export the user details.

Infrastructure Setup:

Create an IAM user, give permission policies for creating EC2, VPC, Subnets, Route tables, Security groups and export all the access keys.

export AWS_ACCESS_KEY_ID=key

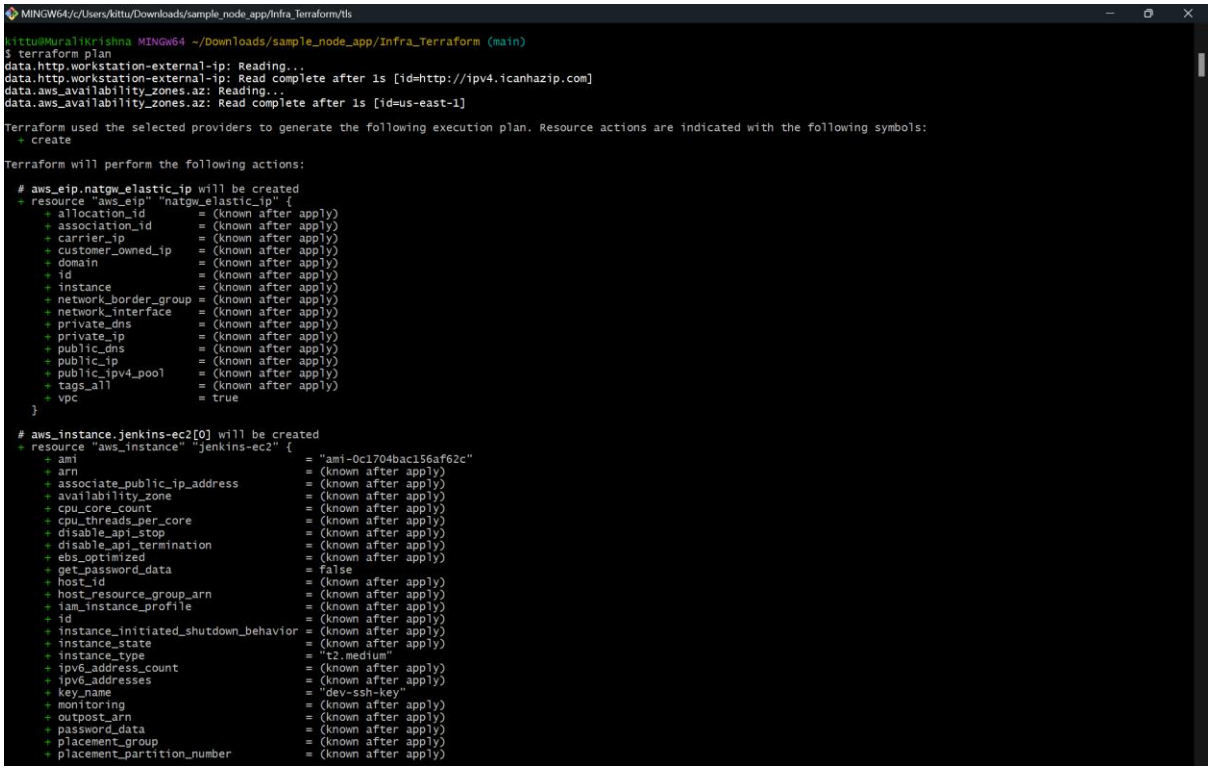
export AWS_SECRET_ACCESS_KEY=key

Now run the Terraform commands

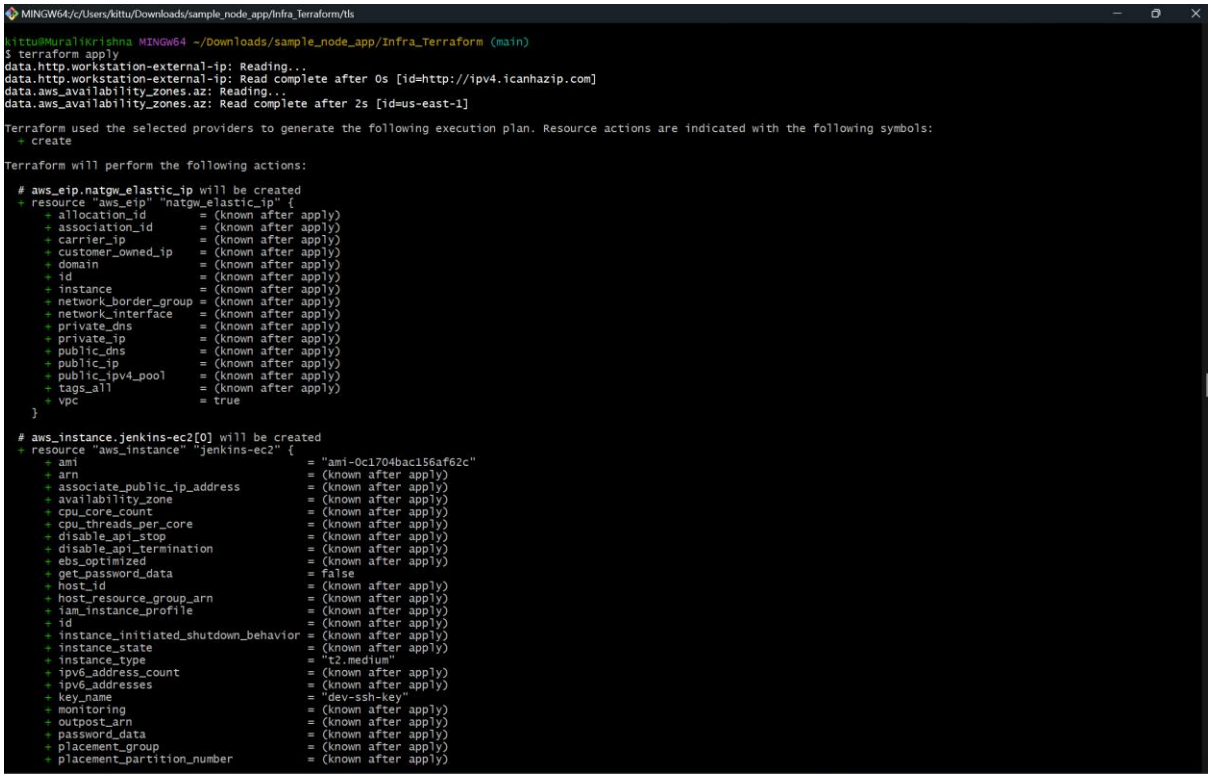
1.terraform init

```
kittu@MuraliKrishna MINGW64 ~/Downloads/sample_node_app/Infra_Terraform (main)
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/local from the dependency lock file
- Reusing previous version of hashicorp/http from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Reusing previous version of hashicorp/tls from the dependency lock file
- Using previously-installed hashicorp/local v2.5.1
- Using previously-installed hashicorp/http v3.4.2
- Using previously-installed hashicorp/aws v4.67.0
- Using previously-installed hashicorp/tls v4.0.5

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

2.terraform plan



3. terraform apply

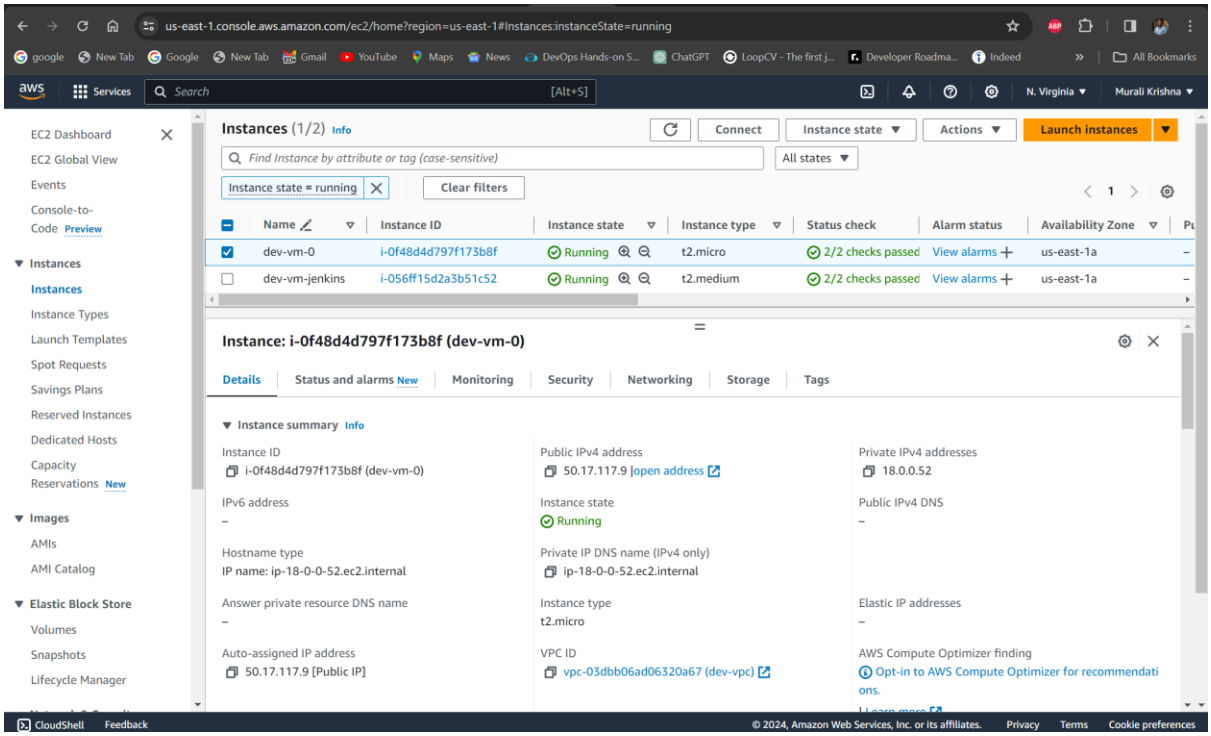


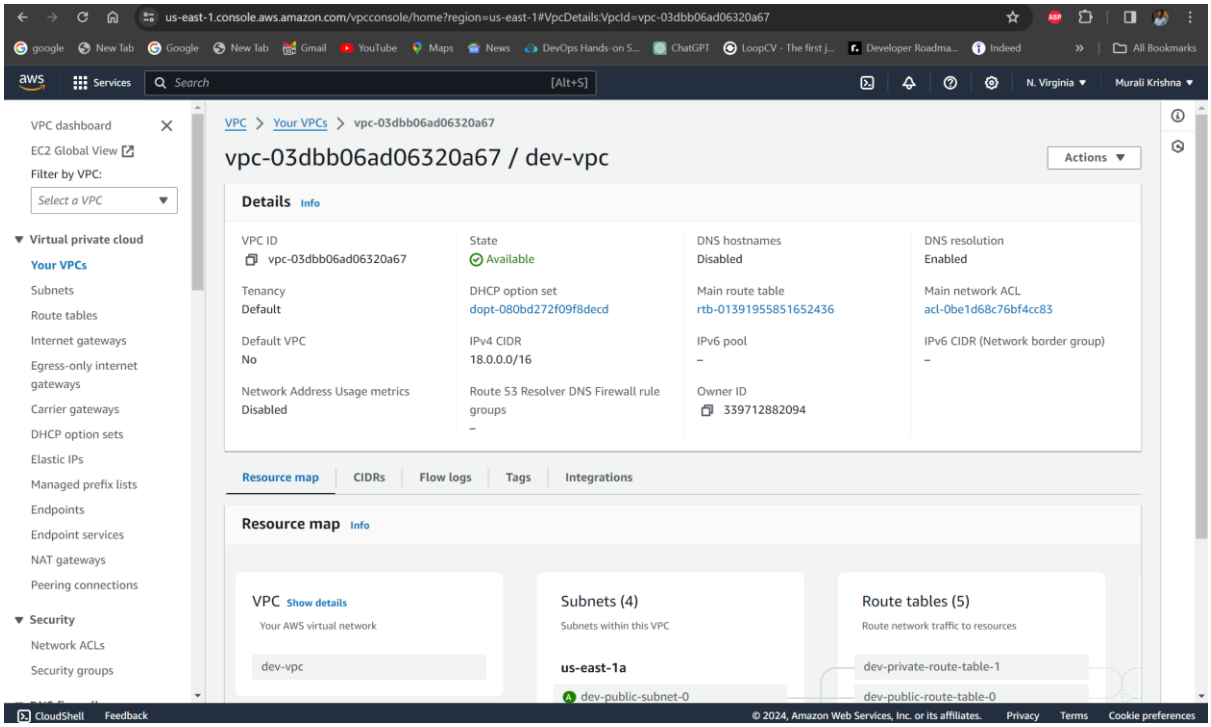
After the execution of terraform apply command, the infrastructure is created and you can find the I.P’s of the EC2 that were created.

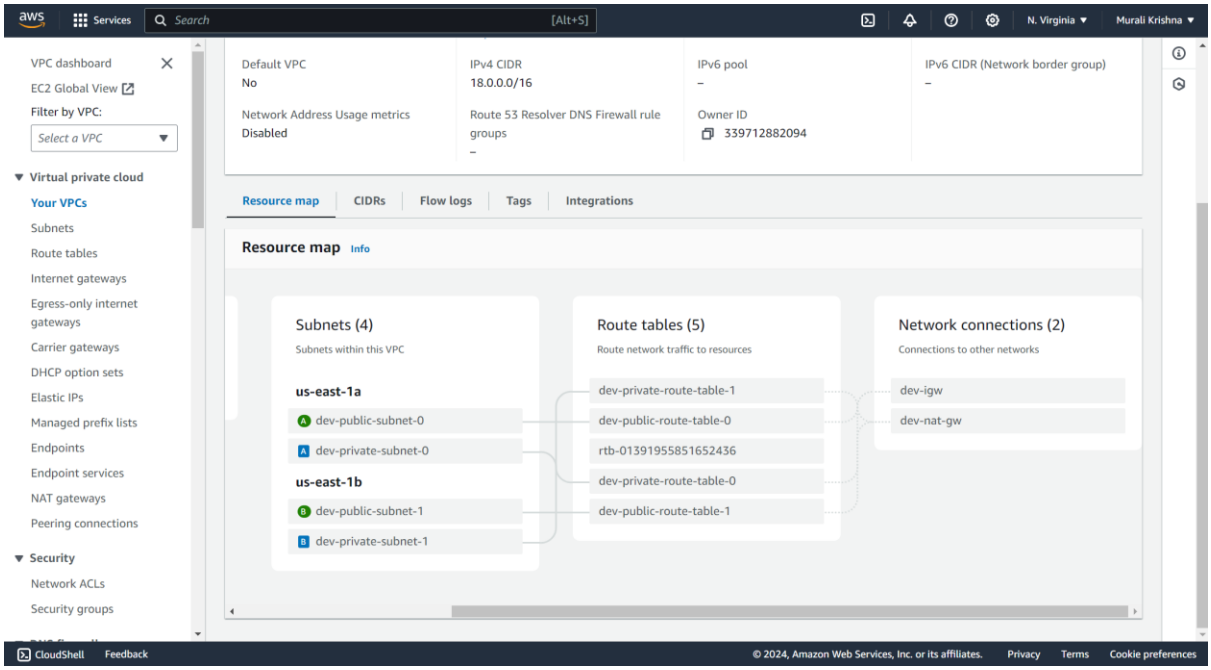
```
MINGW64:/c/Users/kittu/Downloads/sample_node_app/Infra_Terraform/tls
aws_instance.public-ec2[0]: Creating...
aws_route_table_association.rta-sub[0]: Creation complete after 1s [id=rtbassoc-092f05efcc02567b0]
aws_route_table_association.rta-sub[1]: Creation complete after 1s [id=rtbassoc-00f36befaabd2a38a]
aws_nat_gateway.nat_gw: Still creating... [10s elapsed]
aws_instance.jenkins-ec2[0]: Still creating... [10s elapsed]
aws_instance.public-ec2[0]: Still creating... [10s elapsed]
aws_instance.public-ec2[0]: Still creating... [20s elapsed]
aws_nat_gateway.nat_gw: Still creating... [20s elapsed]
aws_instance.jenkins-ec2[0]: Still creating... [20s elapsed]
aws_instance.public-ec2[0]: Creation complete after 26s [id=i-0f48d4d797f173b8f]
aws_instance.jenkins-ec2[0]: Still creating... [30s elapsed]
aws_nat_gateway.nat_gw: Still creating... [30s elapsed]
aws_nat_gateway.nat_gw: Still creating... [40s elapsed]
aws_instance.jenkins-ec2[0]: Still creating... [40s elapsed]
aws_instance.jenkins-ec2[0]: Creation complete after 47s [id=i-056ff15d2a3b51c52]
aws_nat_gateway.nat_gw: Still creating... [50s elapsed]
aws_nat_gateway.nat_gw: Still creating... [1m0s elapsed]
aws_nat_gateway.nat_gw: Still creating... [1m10s elapsed]
aws_nat_gateway.nat_gw: Still creating... [1m20s elapsed]
aws_nat_gateway.nat_gw: Still creating... [1m30s elapsed]
aws_nat_gateway.nat_gw: Creation complete after 1m39s [id=nat-0eeef88f1efcd4a30]
aws_route.private_nat_gateway[0]: Creating...
aws_route.private_nat_gateway[1]: Creating...
aws_route.private_nat_gateway[0]: Creation complete after 2s [id=r-rtb-04dee9e91d8af96bc1080289494]
aws_route.private_nat_gateway[1]: Creation complete after 2s [id=r-rtb-018c6efed8faf003b1080289494]

  Warning: Deprecated attribute

    on get_external_ip.tf line 6, in locals:
     6:   workstation_external_ip = "${chomp(data.http.workstation-external-ip.body)}/32"

  The attribute "body" is deprecated. Refer to the provider documentation for details.


Apply complete! Resources: 28 added, 0 changed, 0 destroyed.

Outputs:

Jenkins-IP = "54.224.185.16"
ec2_public_ip = [
  "50.17.117.9",
]
local_workstation_ip = "43.227.131.160/32"
private-ec2-ip = []

kittu@MuraliKrishna MINGW64 ~/Downloads/sample_node_app/Infra_Terraform (main)
$ ls
ec2.tf             outputs.tf          provider.tf        terraform.tfstate         tls/          vpc.tf
get_external_ip.tf private_subnets.tf  public_subnets.tf  terraform.tfstate.backup  variables.tf

kittu@MuraliKrishna MINGW64 ~/Downloads/sample_node_app/Infra_Terraform (main)
$ cd tls

kittu@MuraliKrishna MINGW64 ~/Downloads/sample_node_app/Infra_Terraform/tls (main)
$ ls
private.pem

kittu@MuraliKrishna MINGW64 ~/Downloads/sample_node_app/Infra_Terraform/tls (main)
$ cat private.pem
-----BEGIN RSA PRIVATE KEY-----
```

In the tls directory, we can find a pem key which can be used for our purpose.

In AWS, we can find all the infrastructure that was created by Terraform. Verify all the services were created.

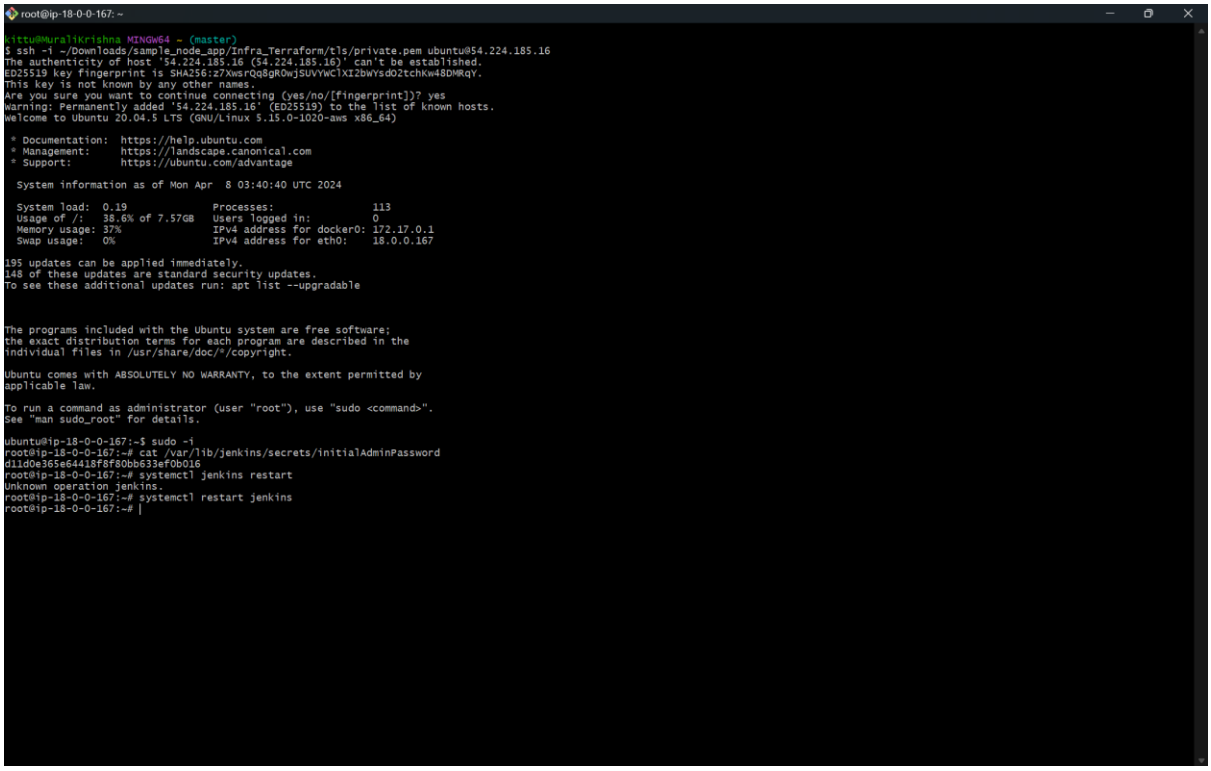




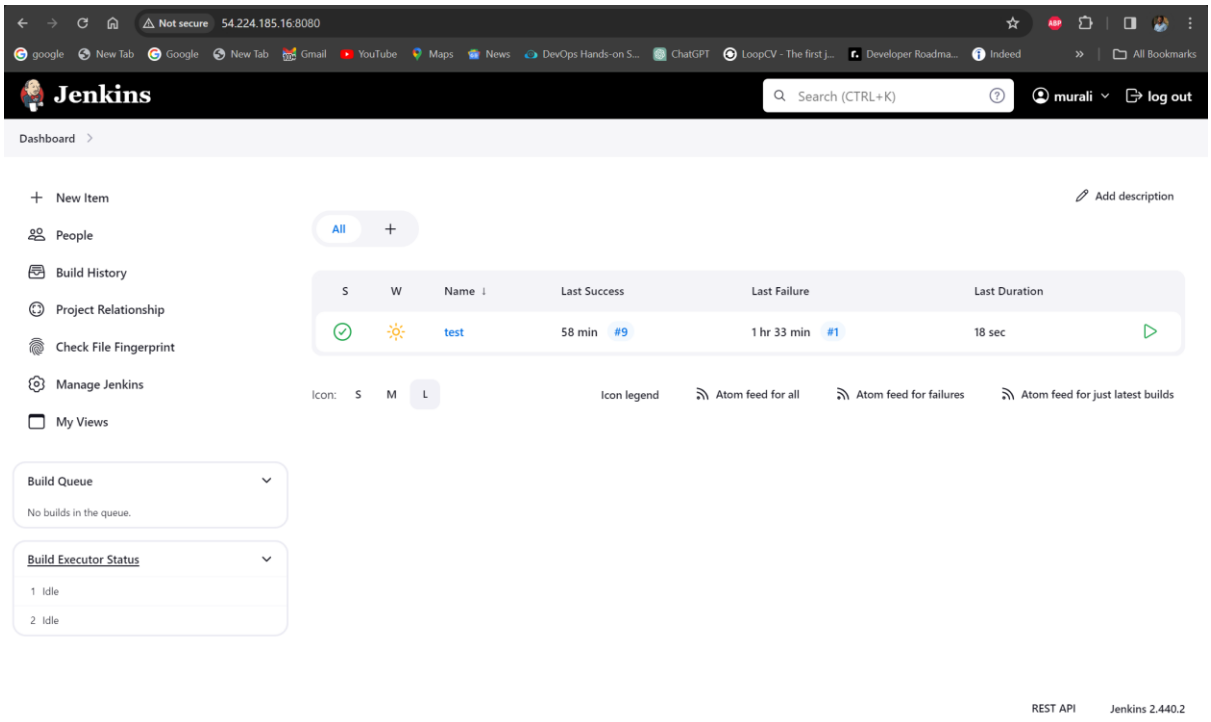


Now, to create the pipeline, log into the EC2 that was created for Jenkins using the pem.key in tls directory.

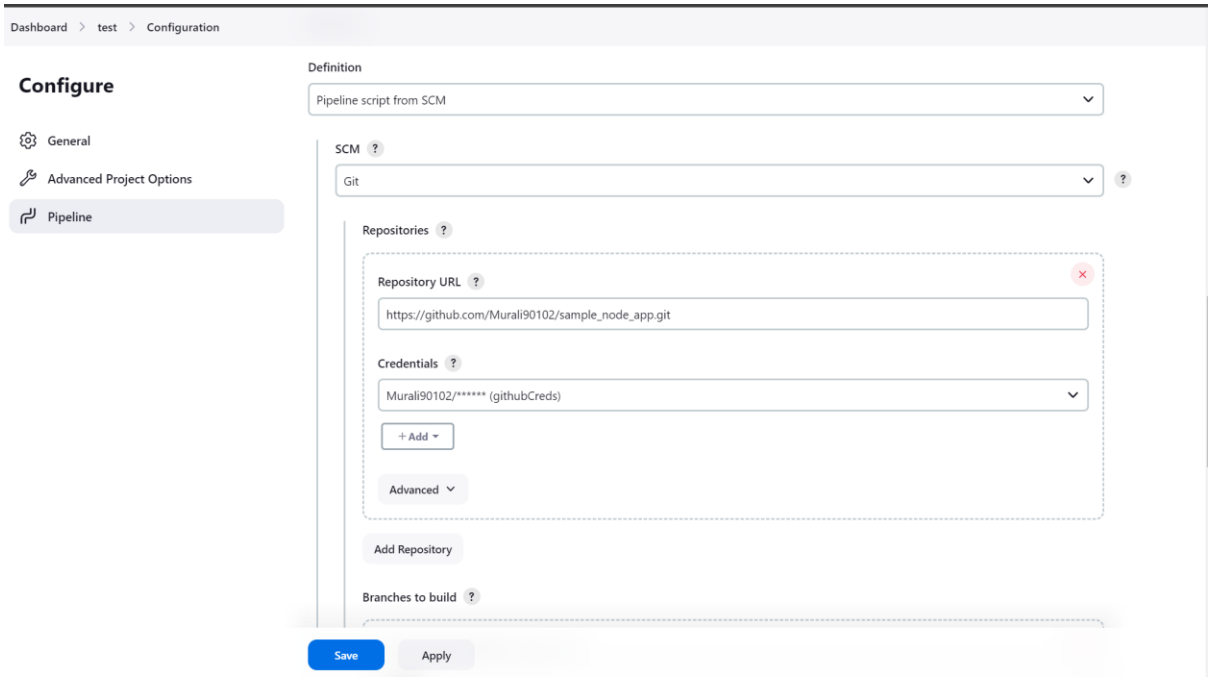
Pipeline Creation:

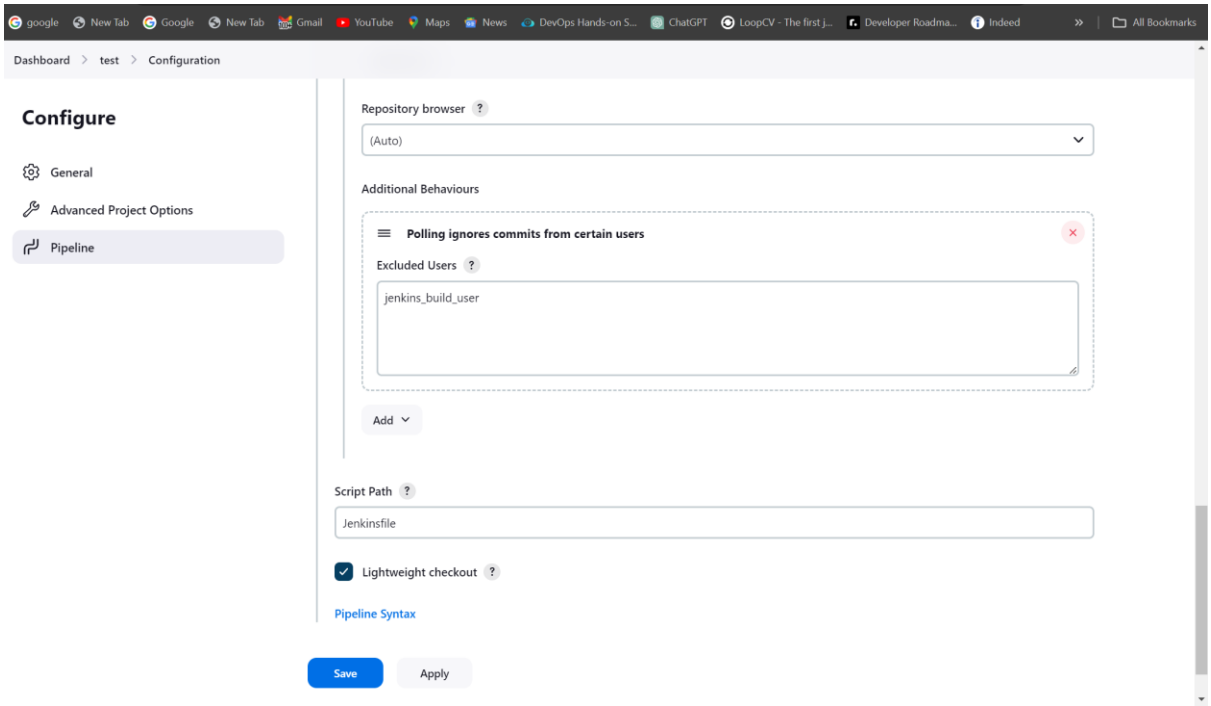


Install the plugins and get into the Jenkins Dashboard.



Create a project and use the Jenkinsfile given in the repository for pipeline script.





Exclude the User jenkins_build_user as the pipeline will create a commit when it passes the docker build stage, it will make changes in the docker-compose file. We don't want the pipeline to get triggered when that commit happens. So, we ignore the commits done by the pipeline which are named under jenkins_build_user.

We want to trigger the pipeline build process only when a certain push is happened in the github repository by a developer or user. So, I checked the GitHub hook trigger for GIT SCM polling and configured it with GitHub webhook.

Steps to create webhook:

1.Open the GitHub repository.

2.Settings➔Webhooks➔Add webhook

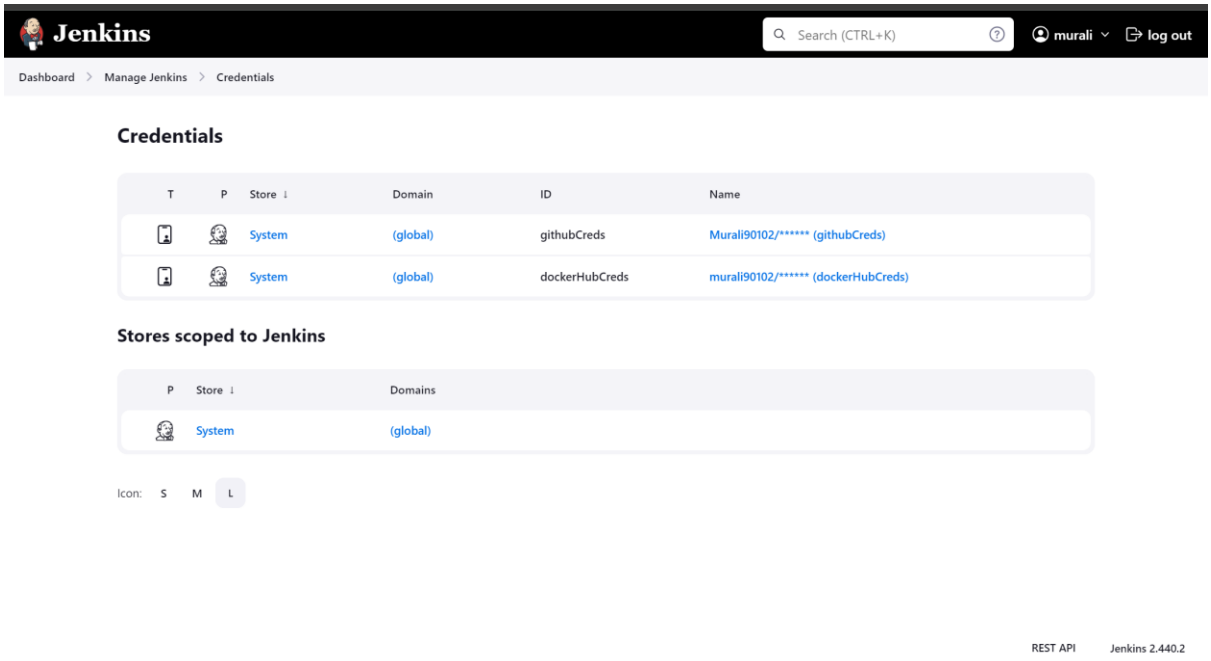
3.Add the Jenkins URL in the payload URL and add /github-webhook/ at the end.

Example: http://54.224.185.16:8080/github-webhook/

This will help to trigger the pipelines in Jenkins when a commit is happened in the GitHub repository.

Configuration of Credentials:

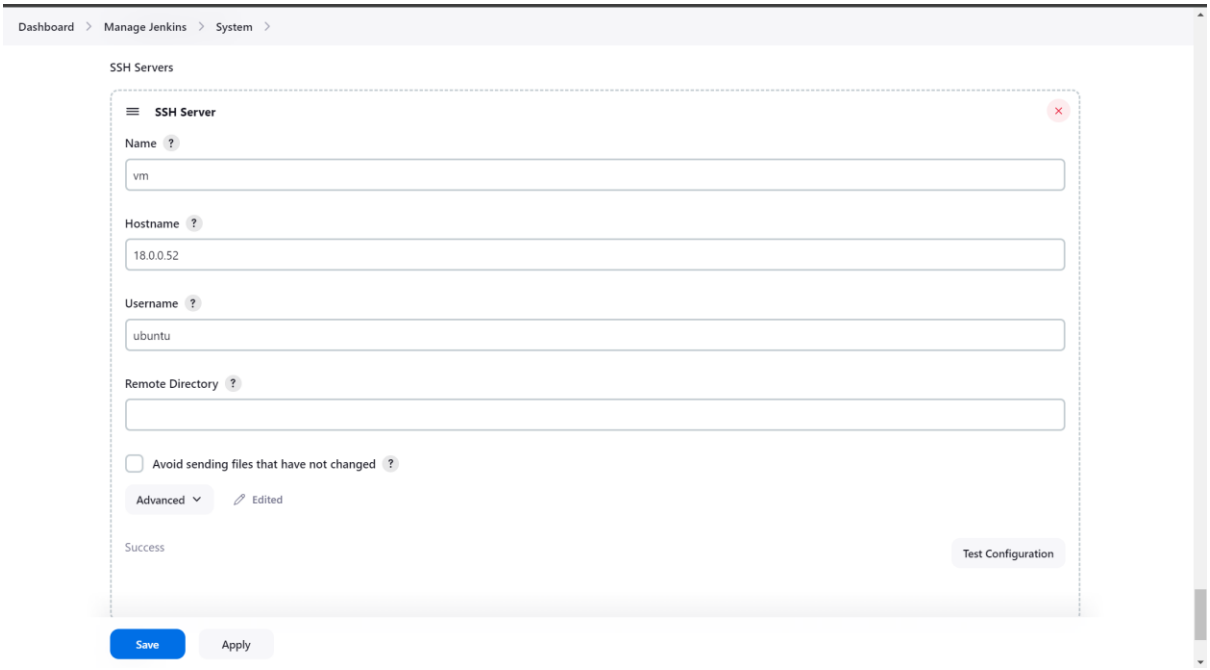
Go back to the Jenkins Dashboard and add the credentials of GitHub and Docker Hub in the Configure Credentials tab with the ID's that are provided in the Jenkins pipeline.



Now, configure the VM to host the application. Install the Publish over SSH plugin in Jenkins.

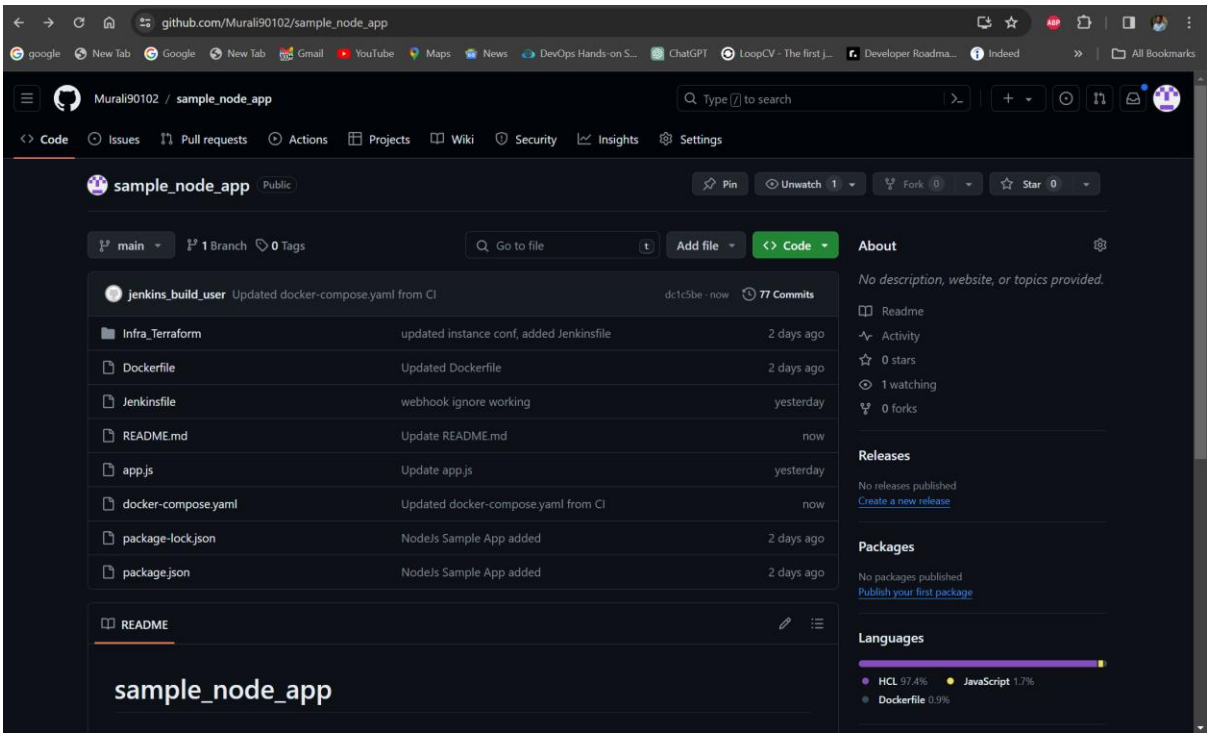
Go to the Jenkins Dashboard➔Manage Jenkins➔System configuration➔System.

Enter the name, private IP, username and password.

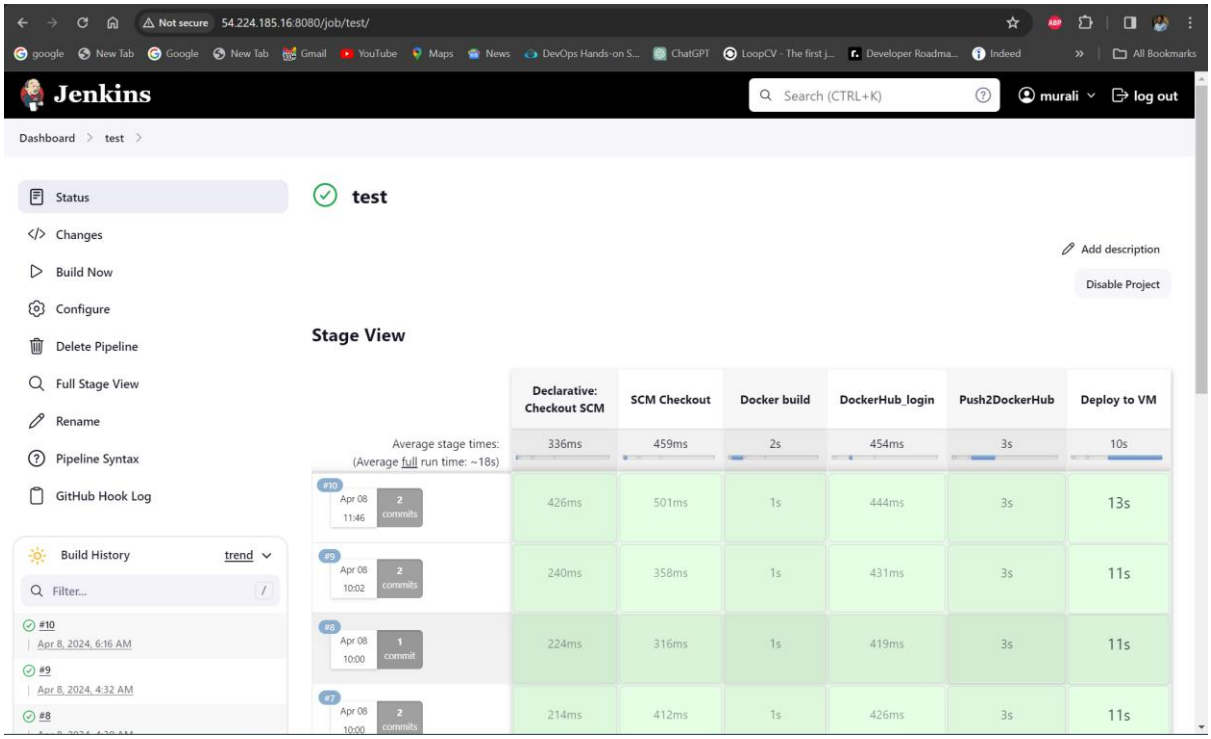


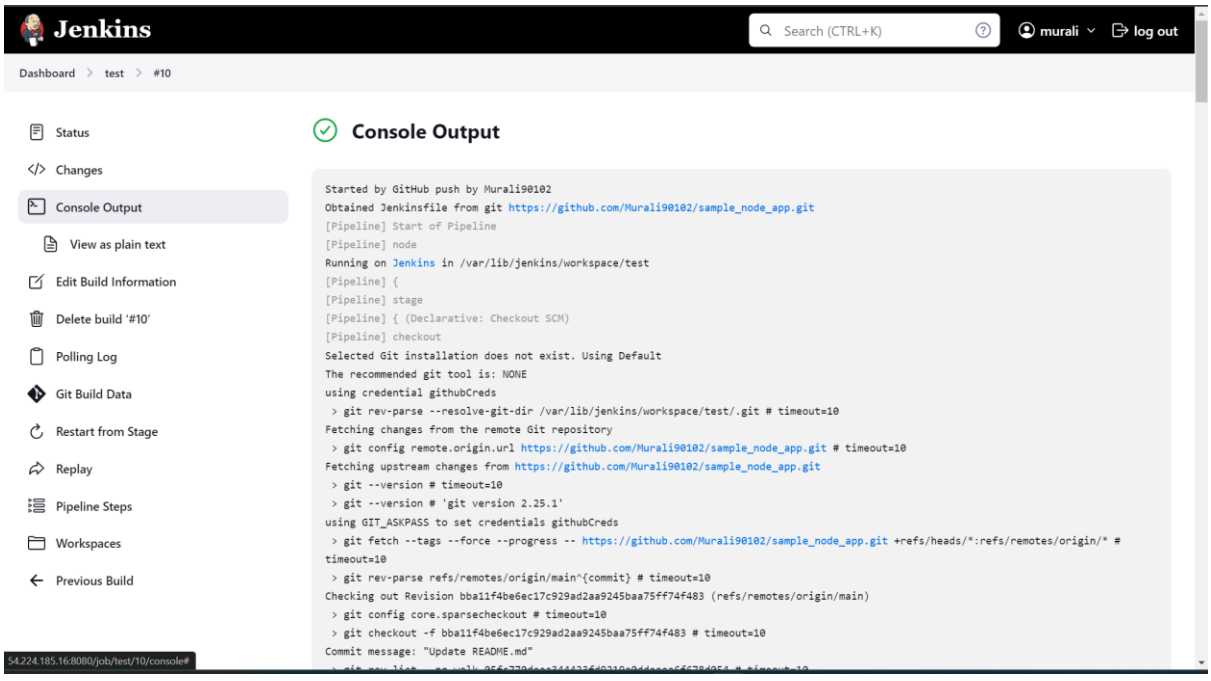
The setup that is required to do the task is configured and ready to go.

I will just commit in the github repo and it will trigger the build pipeline in Jenkins by using GitHub webhook.



I have updated the README.md file and it triggered the pipeline in Jenkins.

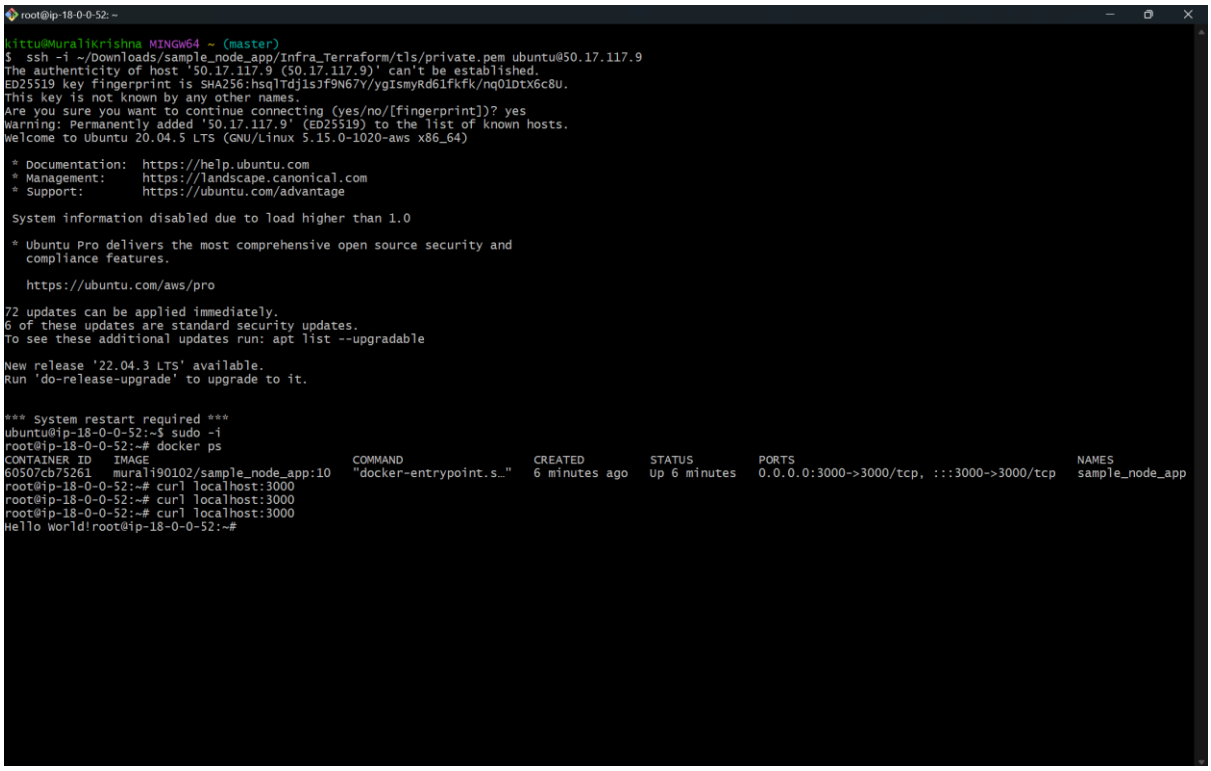




The console output states that pipeline was started by GitHub push. The pipeline passed all the stages and it was SUCCESS.

Deployment verification:

To verify, we'll log into the VM to host the application.



The docker-compose.yaml file has been executed and the docker image has been pulled from the dockerhub. Application is running successfully.

Now, edit the nginx sites-enabled default file with the server details and enable it to access on port 80.

Commands:

1. vi /etc/nginx/sites-enabled/default

   Paste the following in the default file

   server {

   listen 80;

   listen [::]:80 ;

   server_name test.domain_name.in;

   location / {

   # First attempt to serve request as file, then

```
            # as directory, then fall back to displaying a 404.
            proxy_pass http://localhost:3000;
        }
}
```

#Replace the domain with the required name.

2. certbot --nginx -d test.domain.in #replace domain name
3. systemctl restart nginx

The application should be accessible in the browser with the domain name.