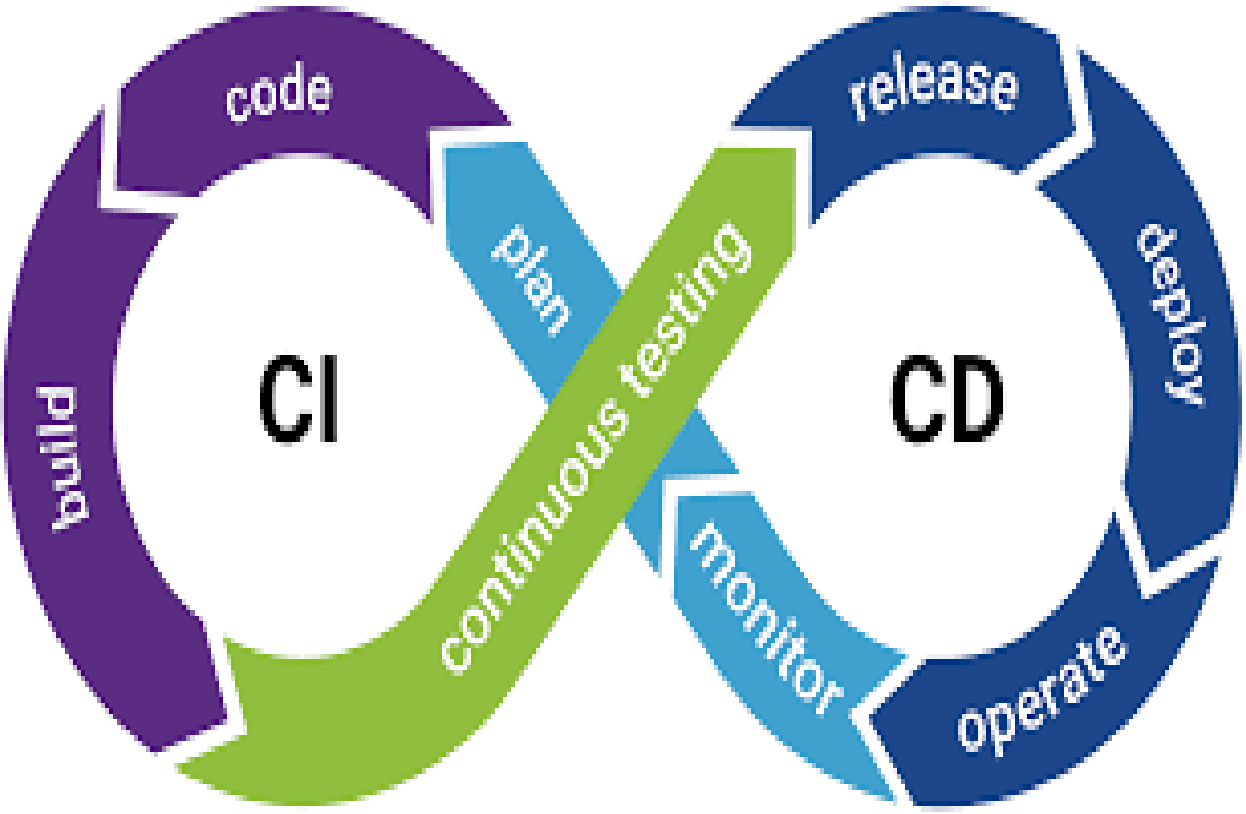
CI/CD Pipeline for Automating Website Deployment



https://www.linkedin.com/in/tejas-pokale-devops/

Tejas POKALE

Server 1 (EC2):

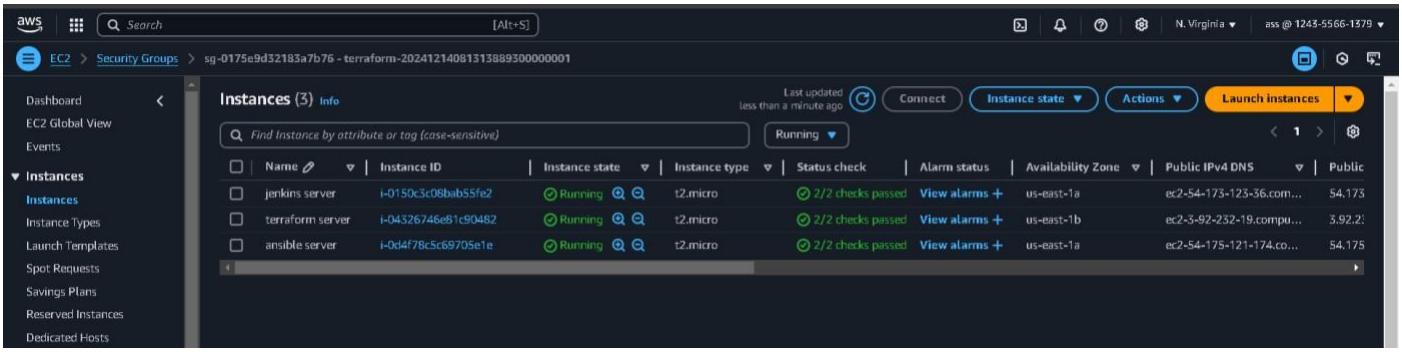
- Purpose: Jenkins server for CI/CD pipeline.
- Role: Automates build and deployment triggers.

Server 2 (EC2):

- Purpose: Ansible server for configuration management.
- Role: Executes playbooks to manage deployments (deployment.yaml, service.yaml).

Server 3 (EC2):

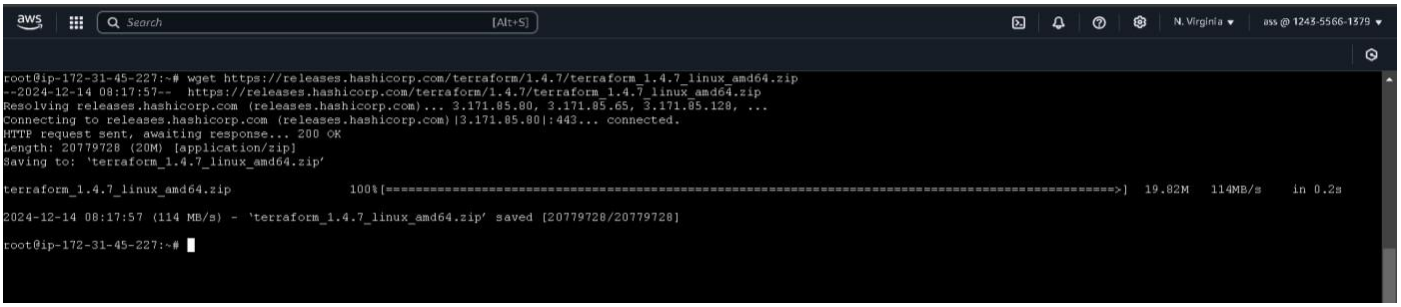
- Purpose: Terraform for infrastructure as code (IaC).
- Role: Manages VPC creation for Server 1 and Server 2.



This terraform installation process:

Switch the normal user to the root user then update my server because I use Ubuntu OS, so OS update requires installing some packages. after installing the terraform and step-by-step type all terraform commands.

The **wget** command is used to copy the terraform package link**.**

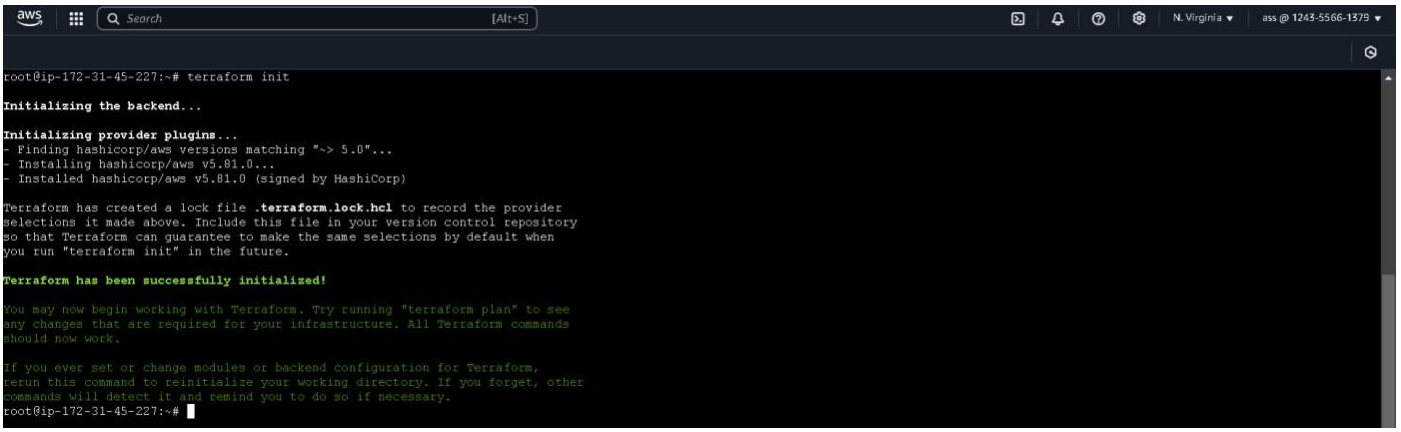


Then unzip the terraform zip file.

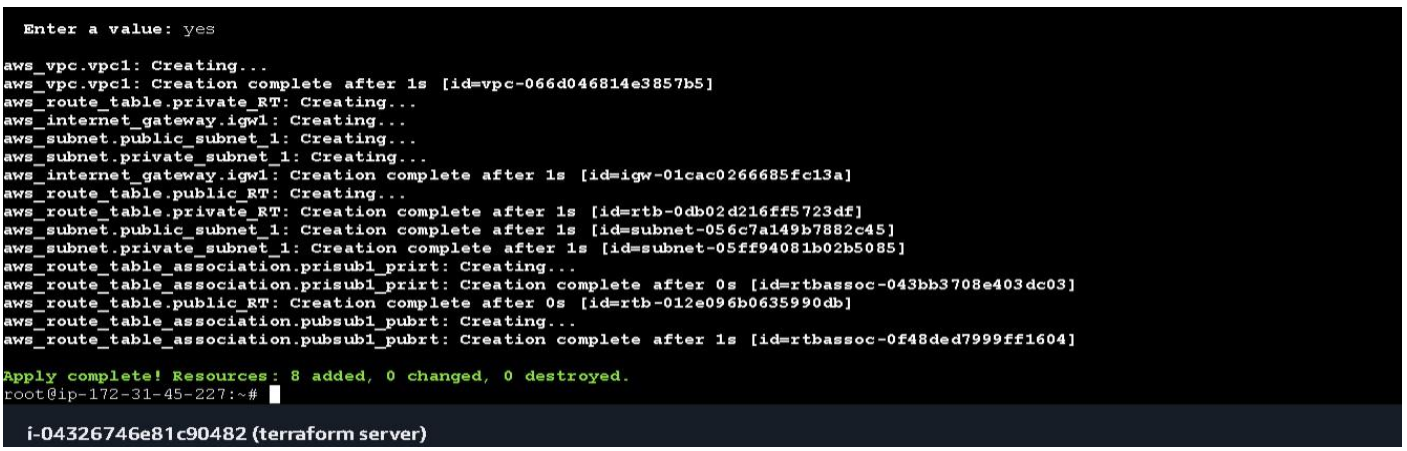
After that unzip that file then run Terraform used to unzip the Terraform file.

The next step is to create a file using with Terraform extension. for example (main.tf)

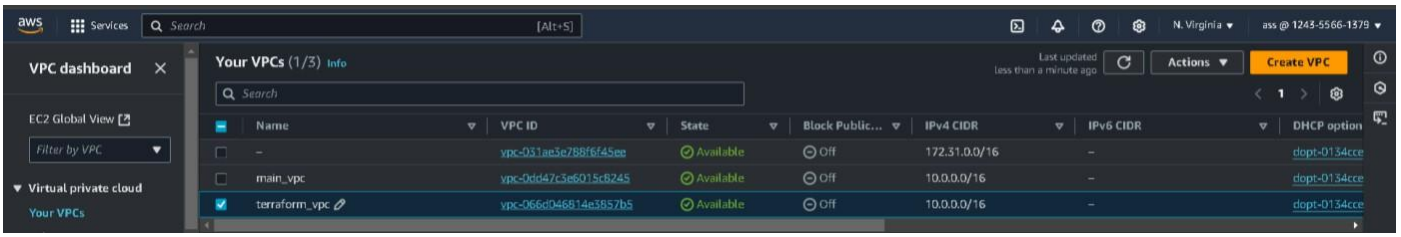
(Terraform init) This command is used to create a terraform repo and also check the terraform code



Step by step type the terraform command and eventually create a vpc for two servers. The second command is used for planning I mean terraform tell a server. what will it create using that code? Last command is terraform apply it command using for run code and then create a vpc.



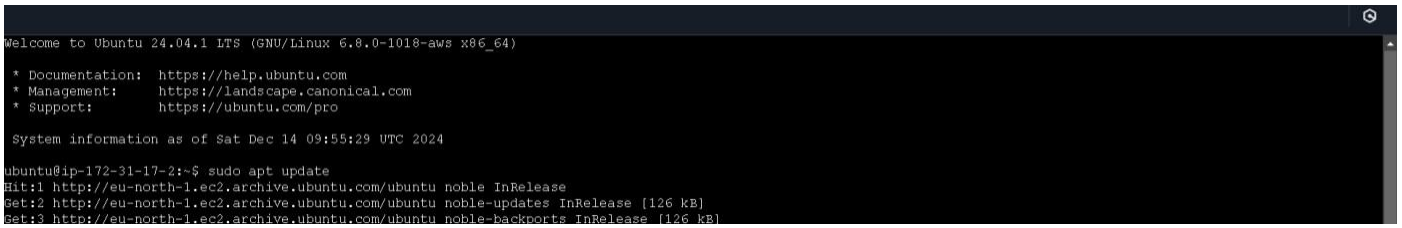
Finally, successfully create a VPC:



JENKINS SERVER:

(Server 1 Jenkins server)

Sudo apt update:



Then install the Jenkins Java package because Jenkins build Java so that Java package requires.

Jenkins install package:

Then go to the Ansible server then install the Ansible package in **(sever 2 ansible server)**

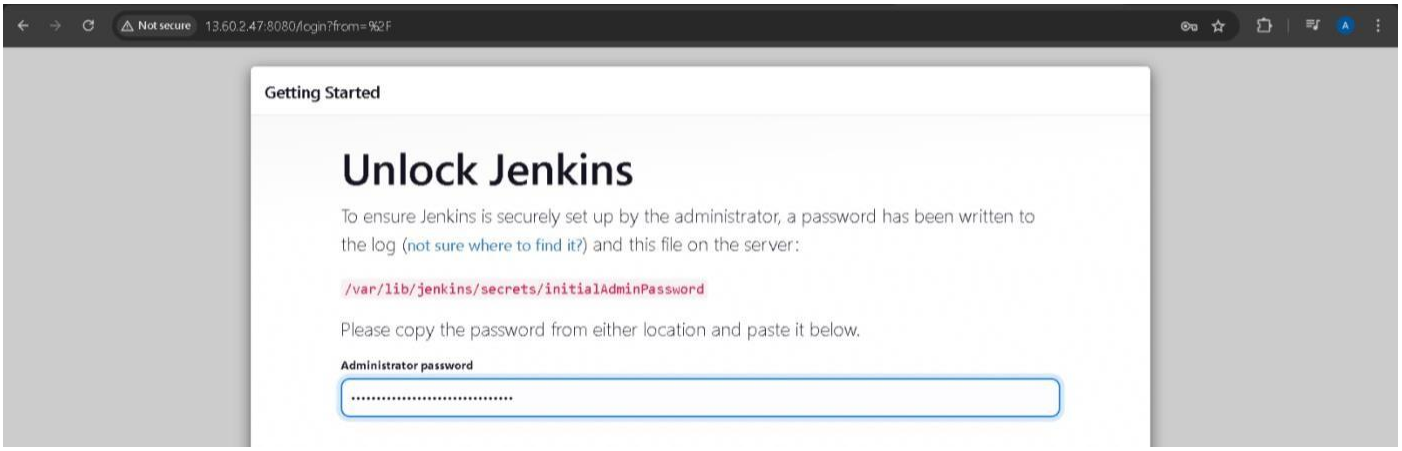
Then install the DOCKER in server 2.

Then check docker is successfully installed in server 2

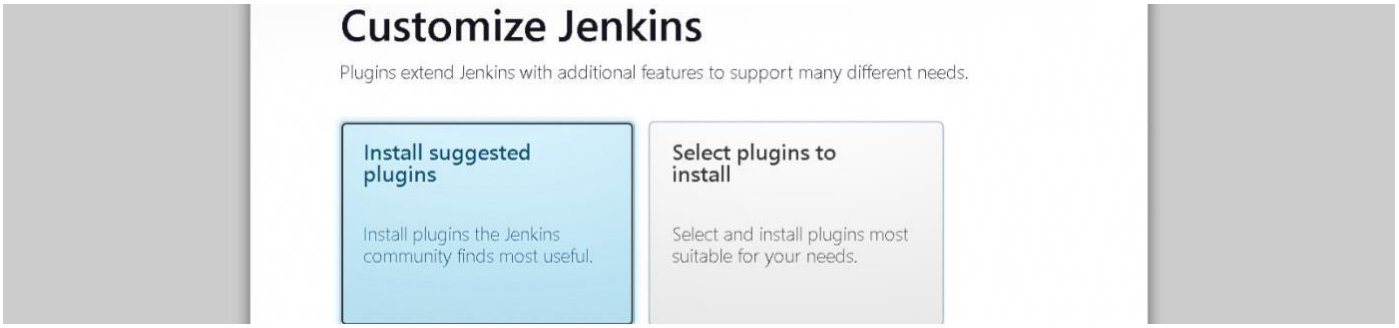
Docker images

Next step: copy the Jenkins server public IP address number and then paste it within the browser. Then open the Jenkins page. Copy the path above the admin password.

1)

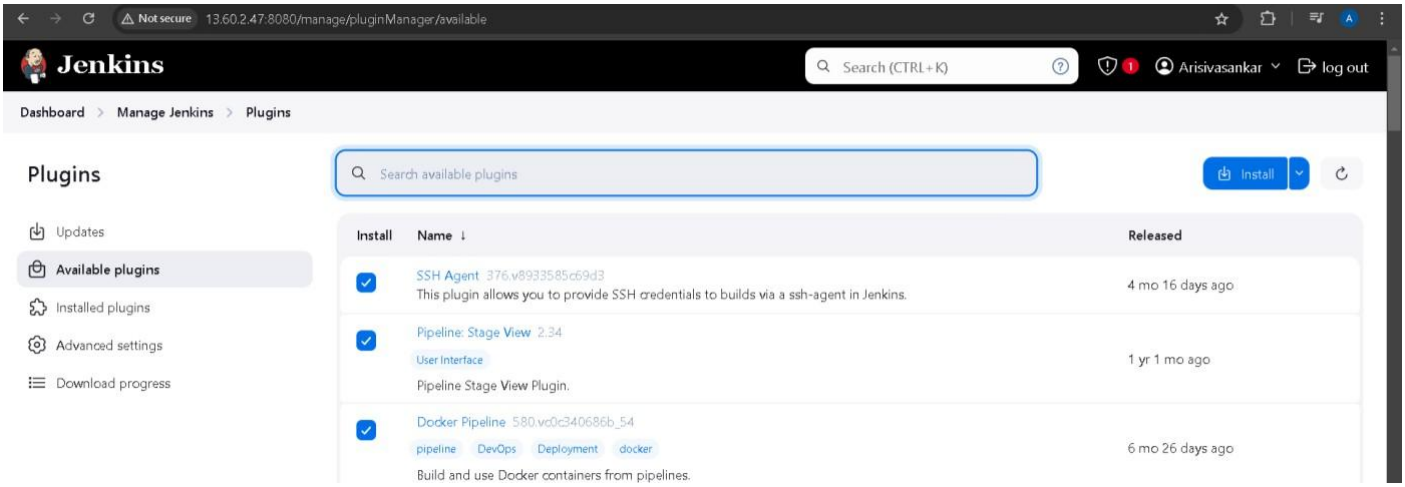


2)



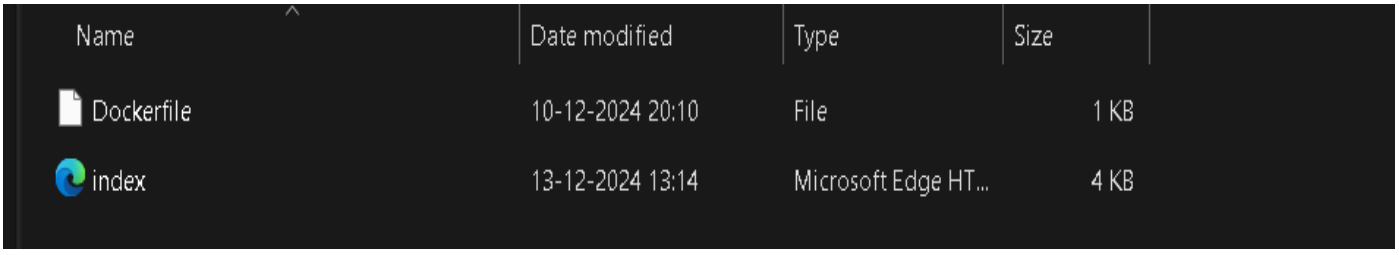
Then create a username and password after install the default plugins.

Install this plugin **(ssh-agent, pipeline stage view, docker pipeline)**



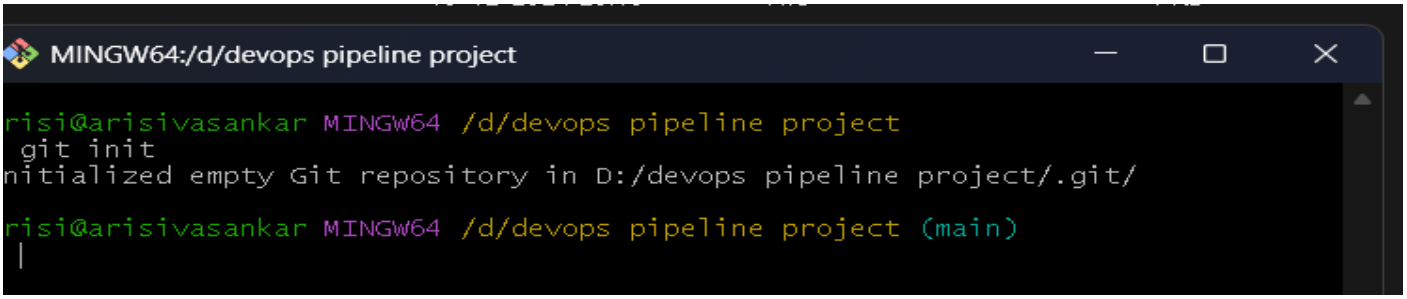
Install important plugins then restart Jenkins and type the username and password.

And then already created a docker file and index.html file then pushed those files to GitHub



GIT:

Git init. This command is used for create a git repo because I didn't create a git repo then I never pushed those files to GitHub.

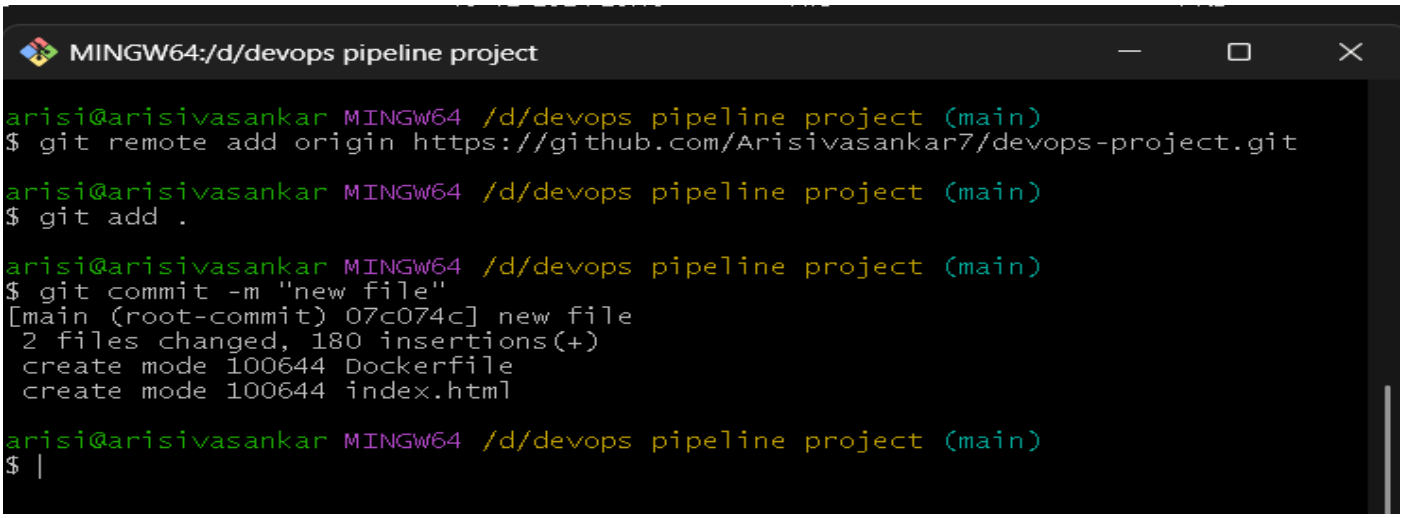


Type the step by step git command.

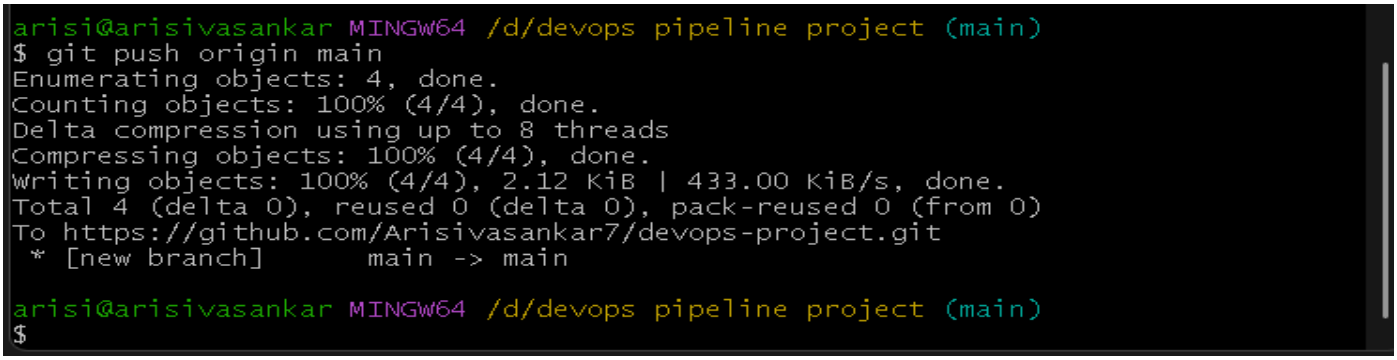
Git remote add origin: This command is used to connect the remote repo.

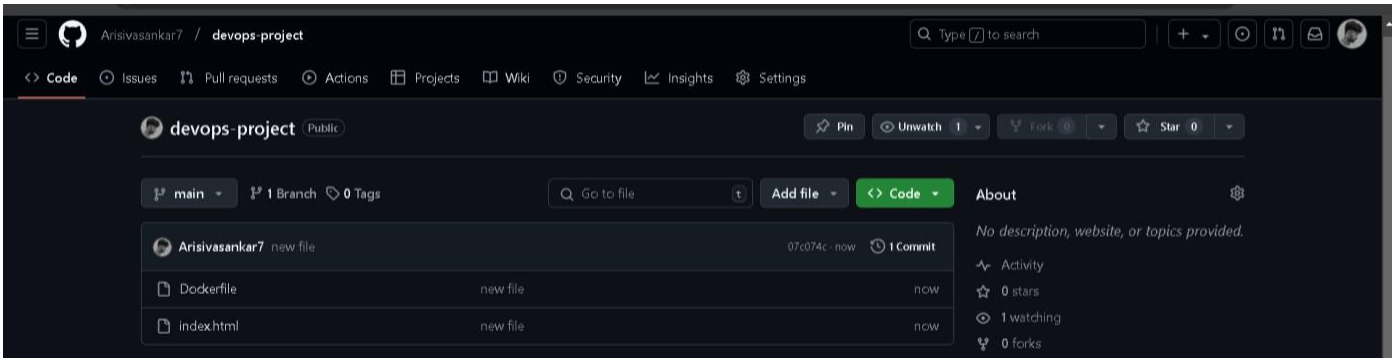
Git add: used to push those files in the staging area which means temporary storage.

Git commit: command used to push stage area to the local repo.

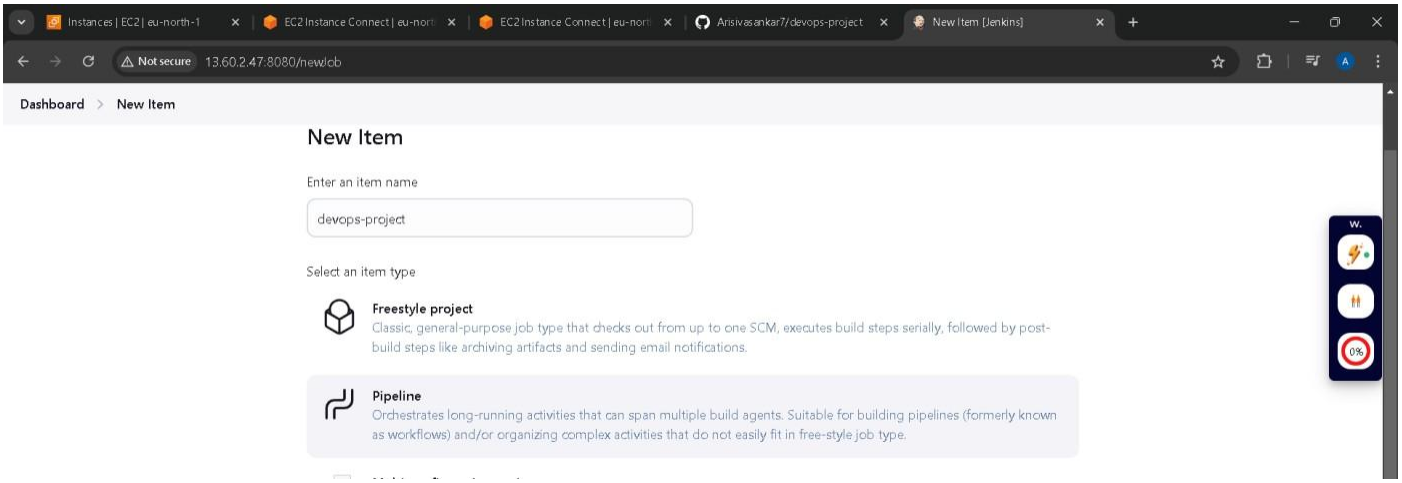


Git push origin main: these commands are used to push those files in the remote repo.

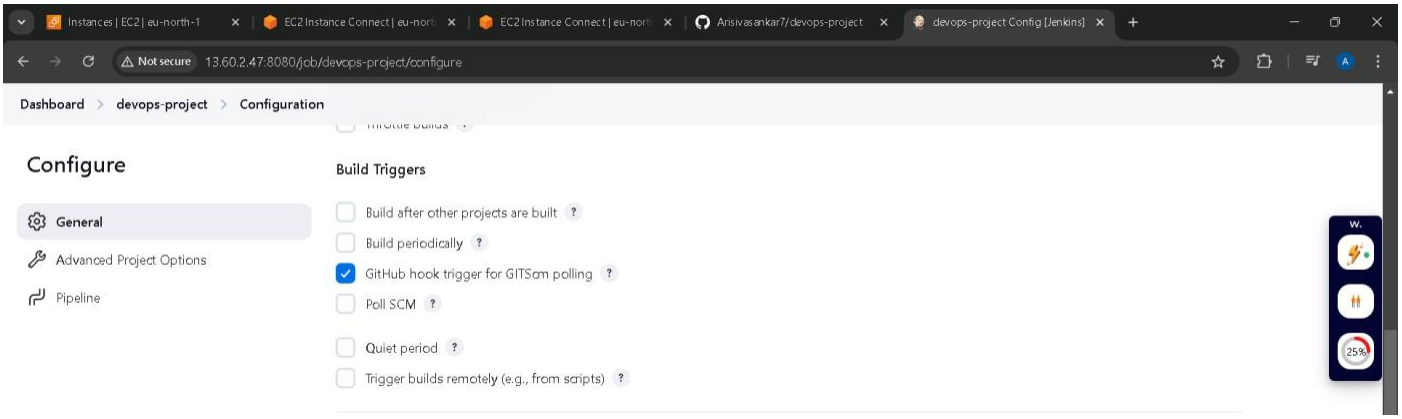




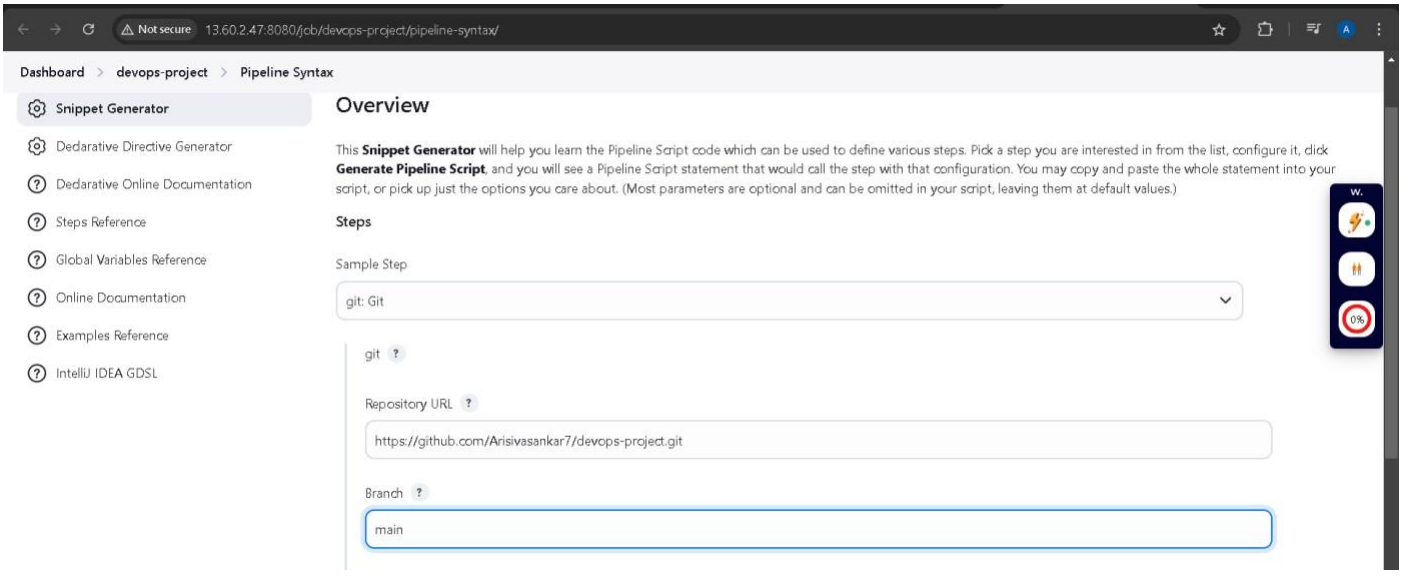
Then go back Jenkins page. Create a new job.

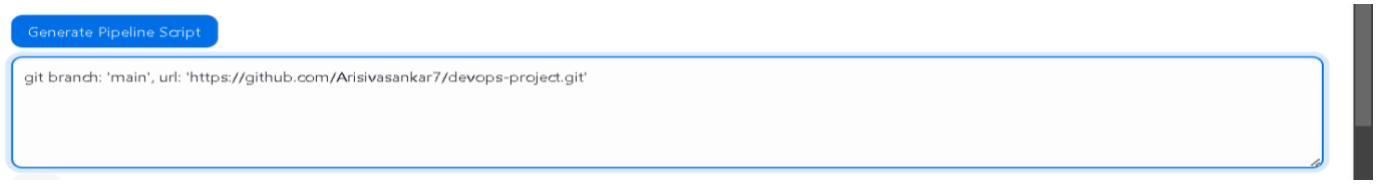


Select the pipeline then click (ok) then choose build trigger.



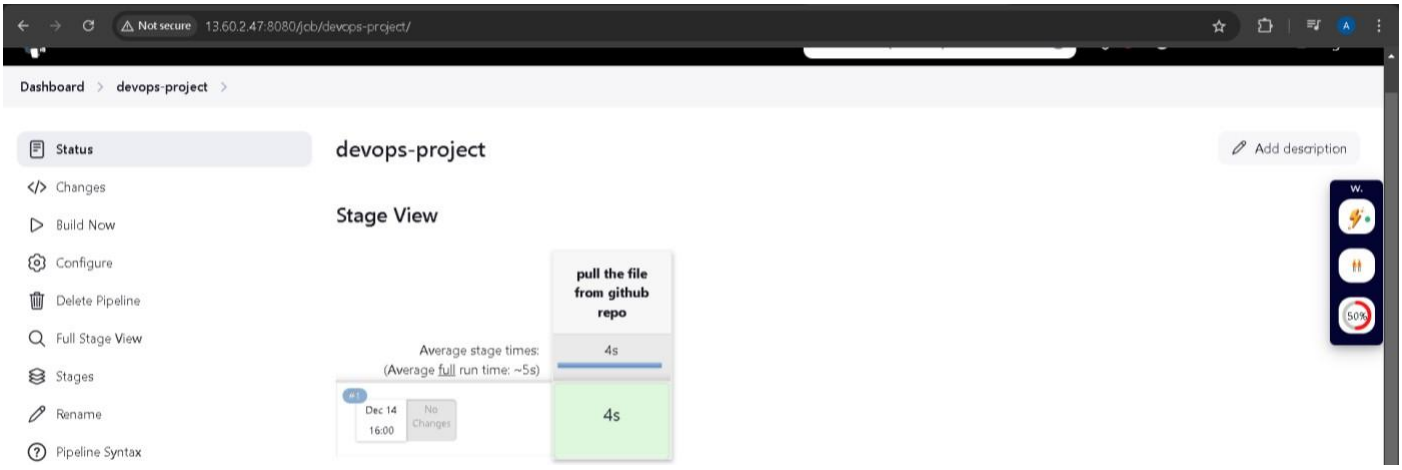
Then click pipeline syntax and choose the **git: Git** and copy the GitHub repo link. After that paste it into the repo URL and select the branch finally click the generate pipeline script and automatically create a script then copy that script and go back script page and paste it.





This script is used to pull the files from a GitHub repo

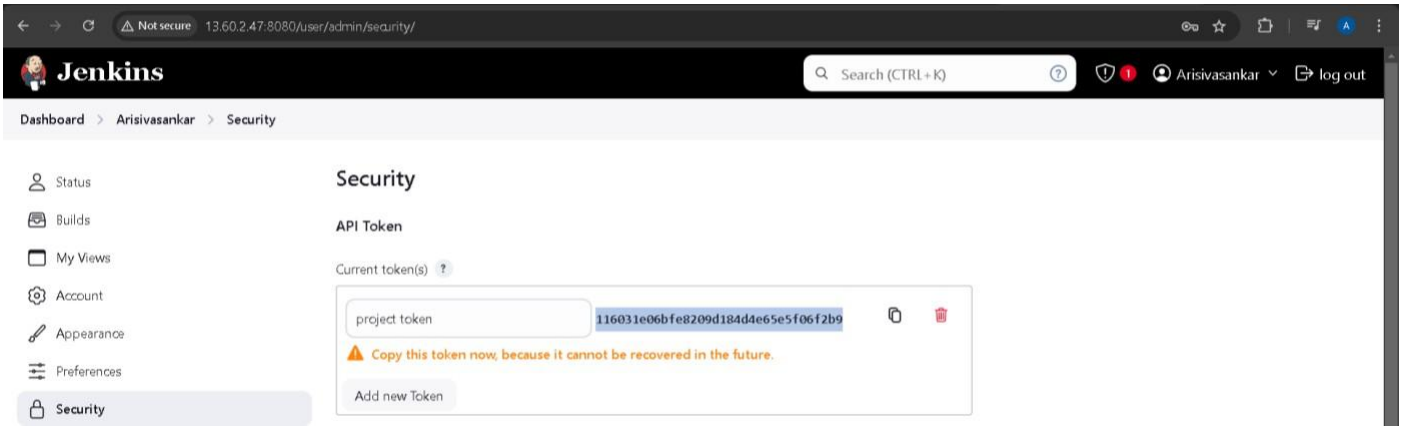
Then click apply and save and click build now then automatically pulled the files from GitHub repo.



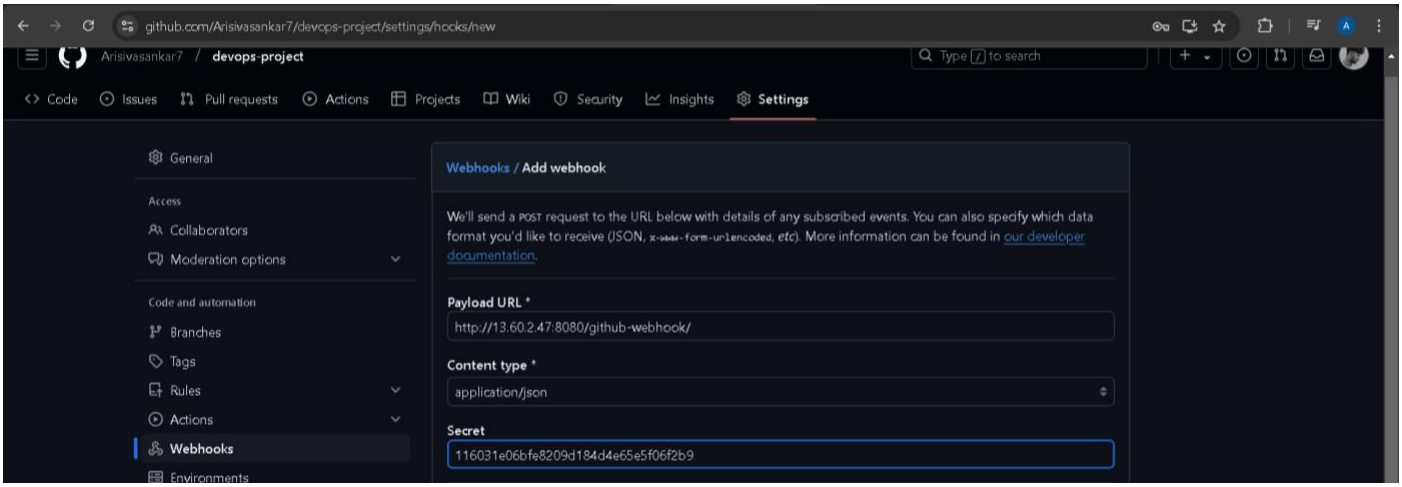
Go back to the Jenkins server and change the Jenkins path.

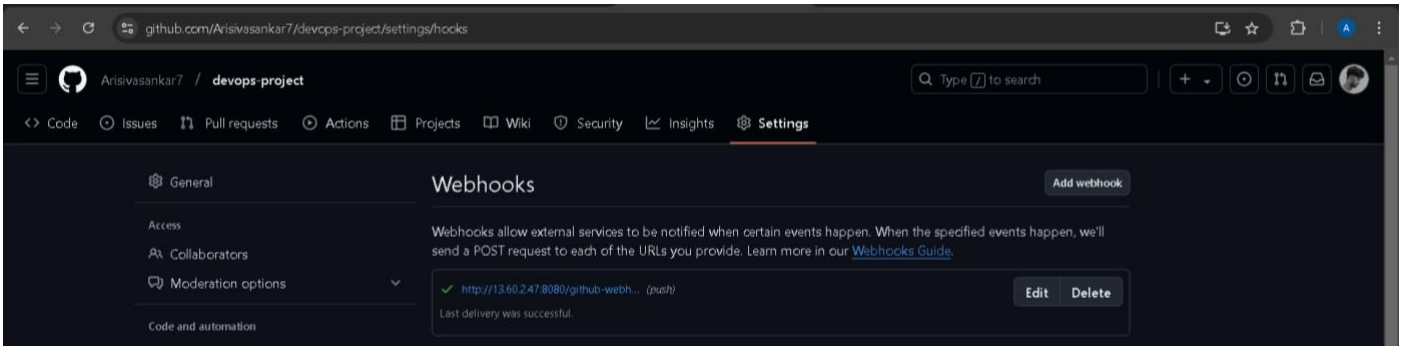
Successfully run first stage script

The next step create a GitHub webhook because if I don't create a GitHub webhook then never automatically triggers the stage script so first create an API token. this token is just for security purposes and to create a GitHub webhook. What it takes to make it create.. the first Jenkins page URL and next is API token.

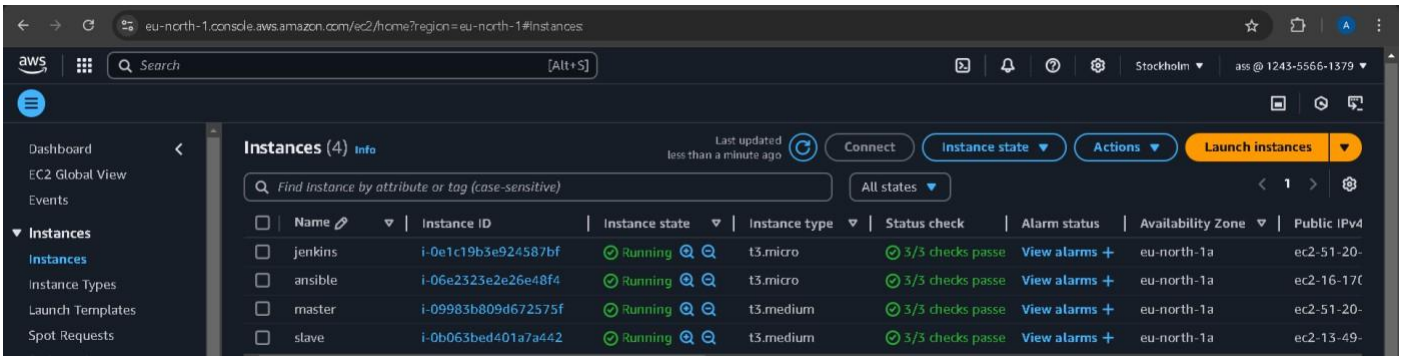


GitHub webhook page:



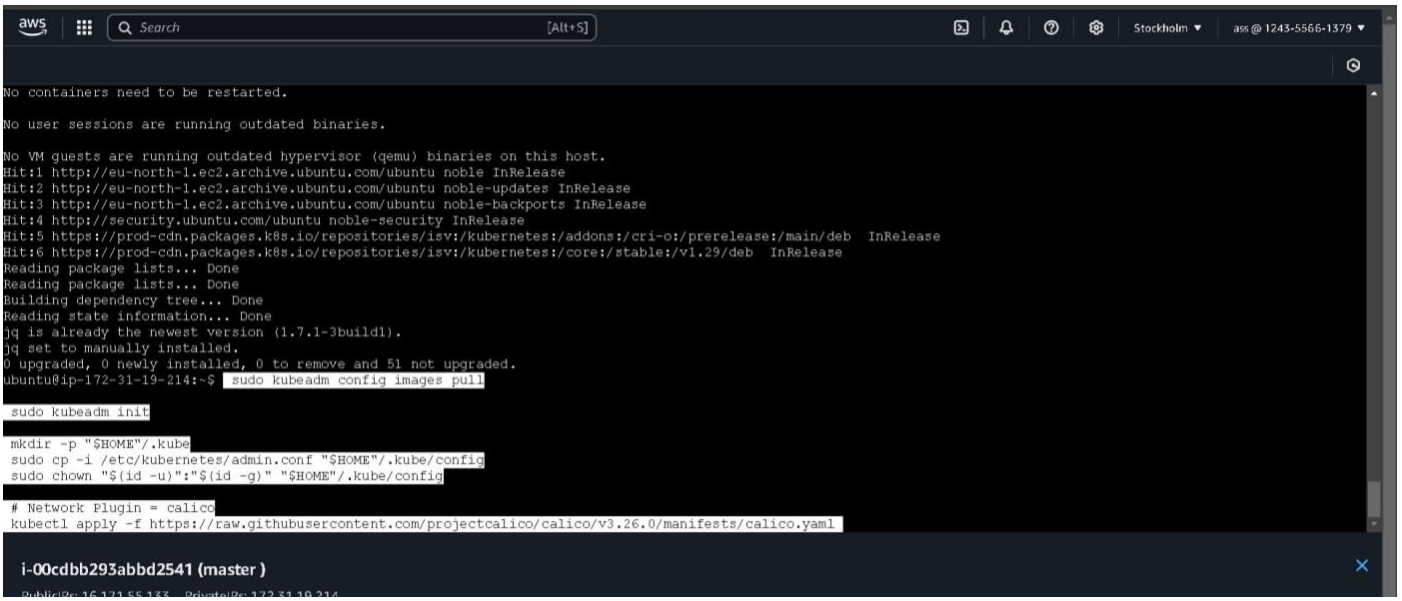


The next concept is to create a master machine and a slave machine:

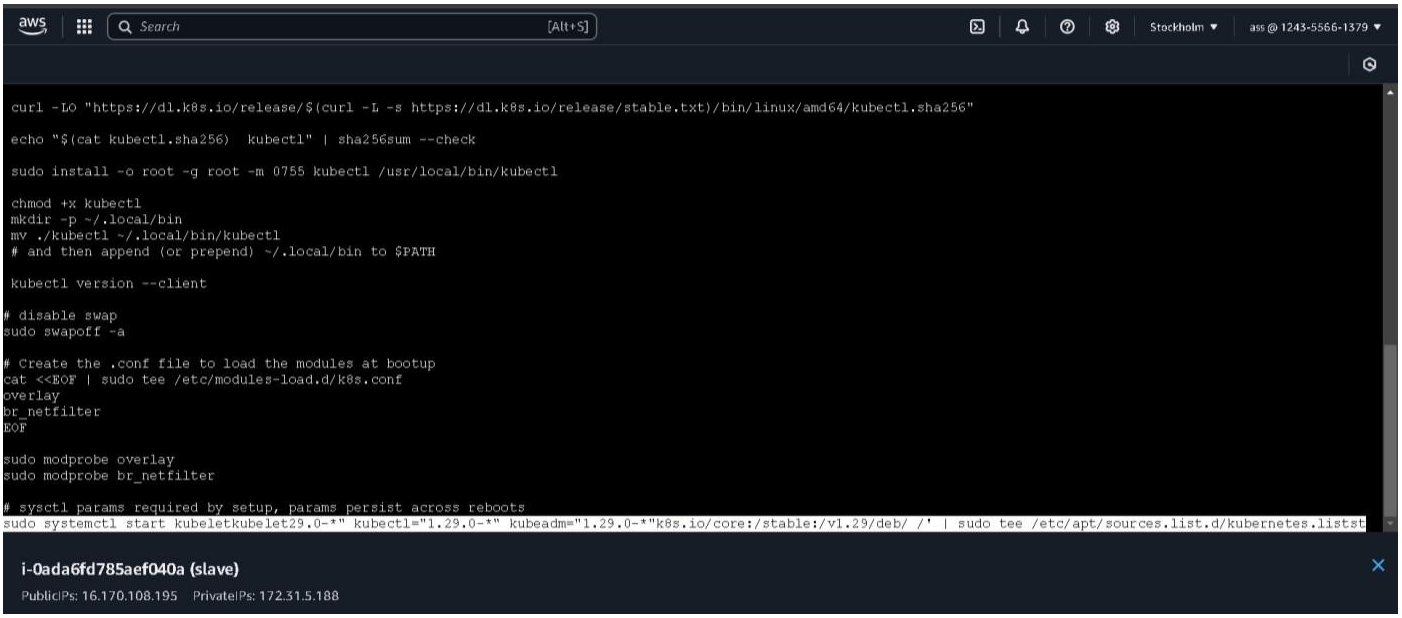


Kubernetes:

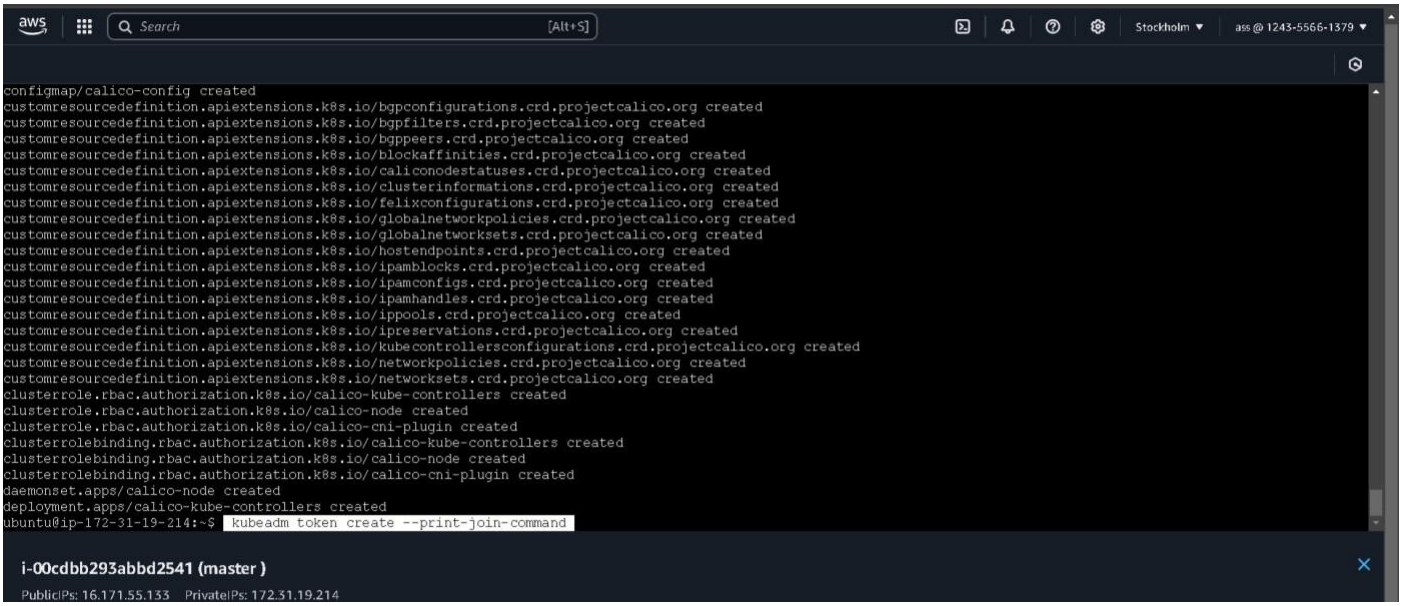
Master machine:



Slave machine:

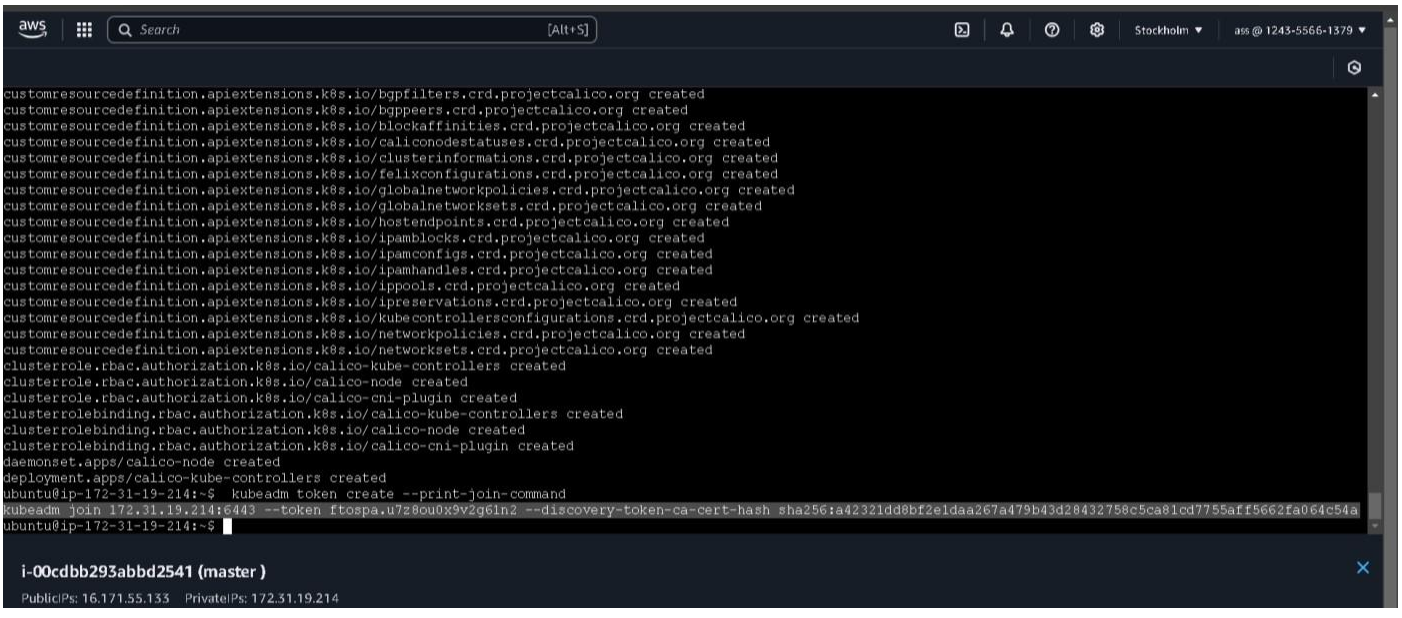


Then copy and paste kubeadm token:



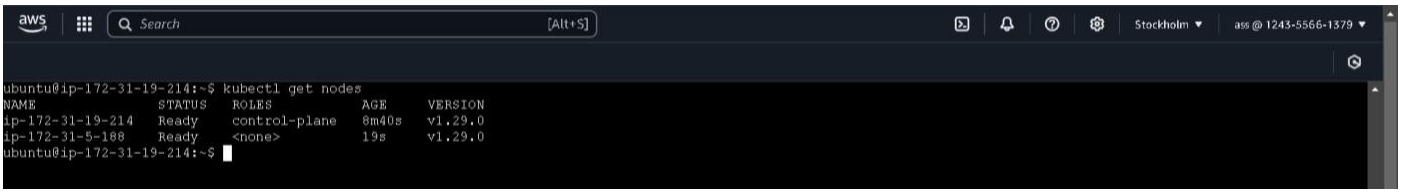
Slave:

Then copy this line and those line paste it slave server.



Successfully configurated kubeadm cluster.

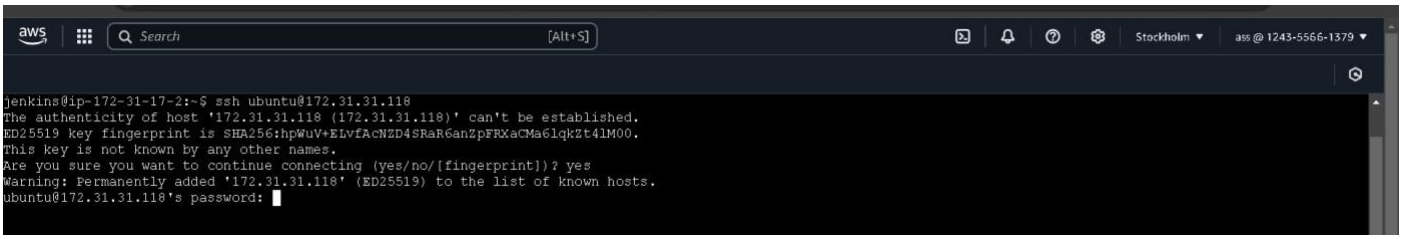
Type this command after successfully congigure kubeadm cluster.

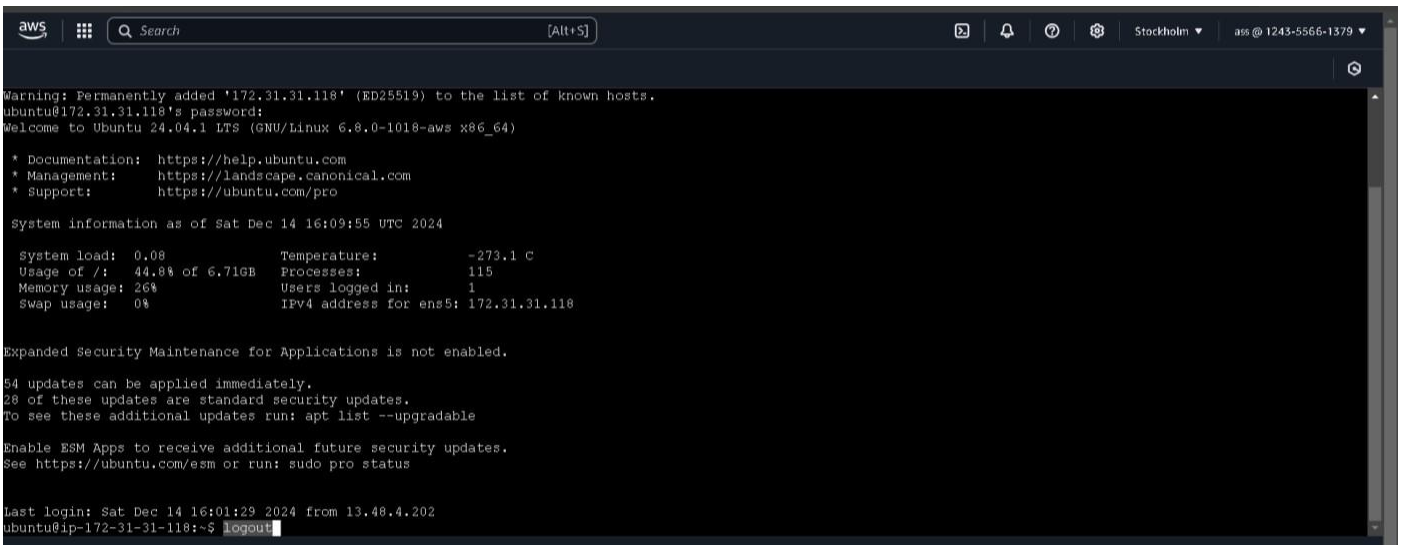


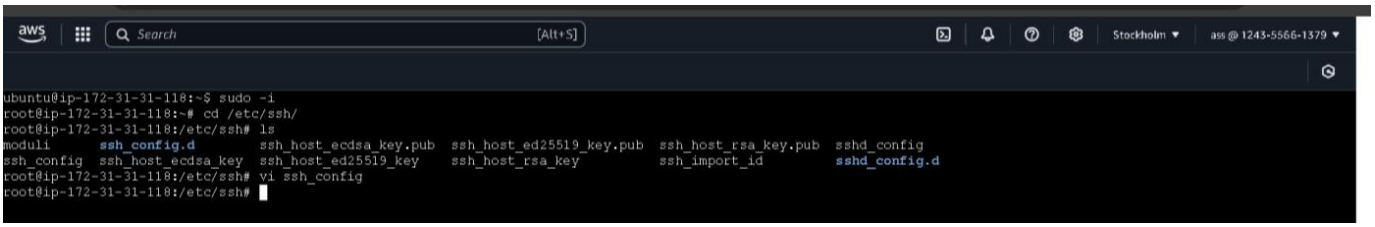
Next setp is passwordless authentication:

First switch normal user to Jenkins user then create a passwordless key. After that go to the ansible serverThen modify some files and successfully modify those files then create a password for security purpose and that password helpful to connect server to server so it requires and also same process in master server.

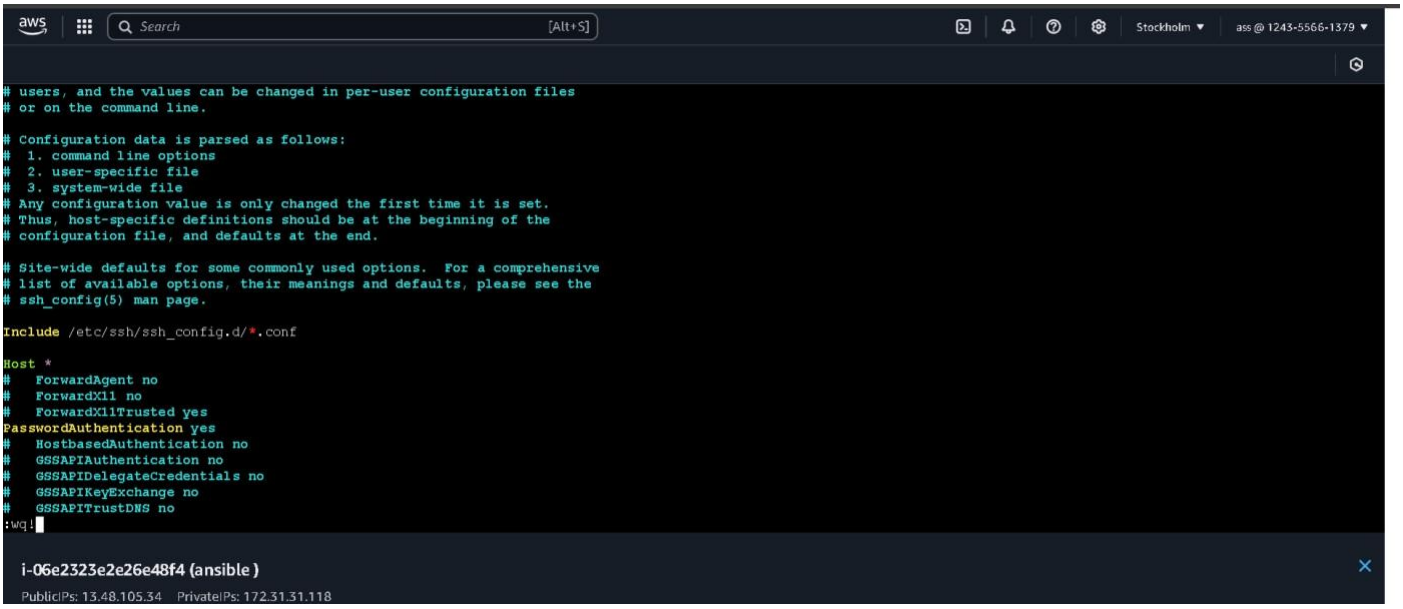
ssh ubuntu@123.34.445.6 : this command is using to connect server to server and it ask password then type it password then connected that server in Jenkins server



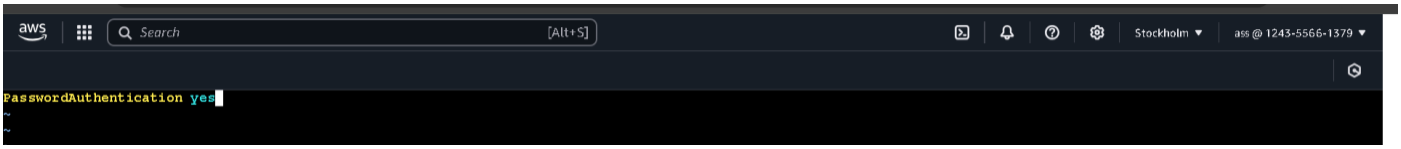




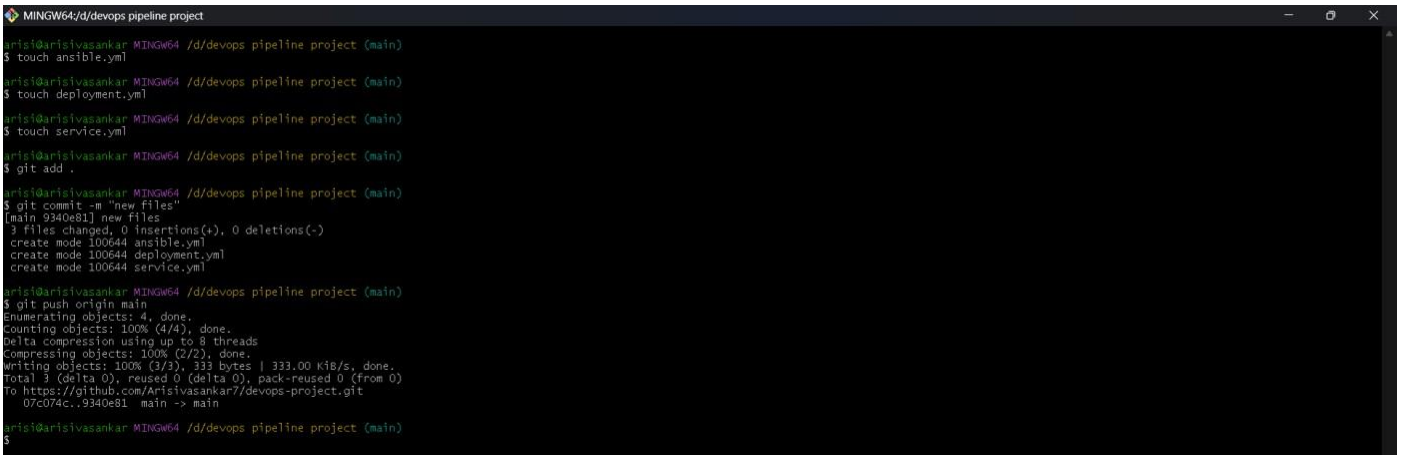
First file



Second file.

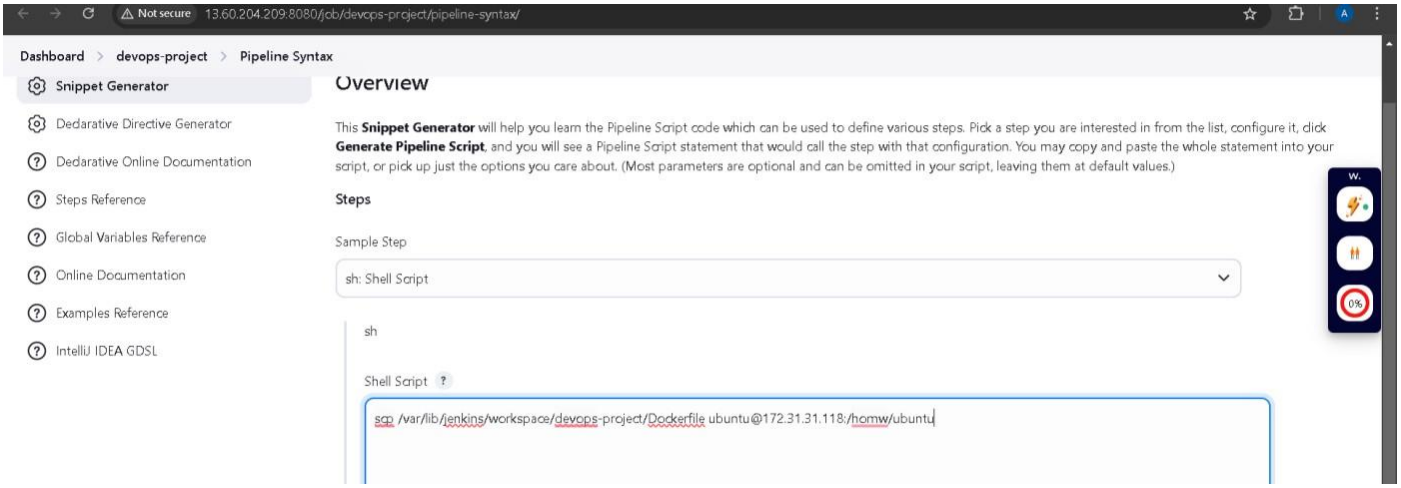


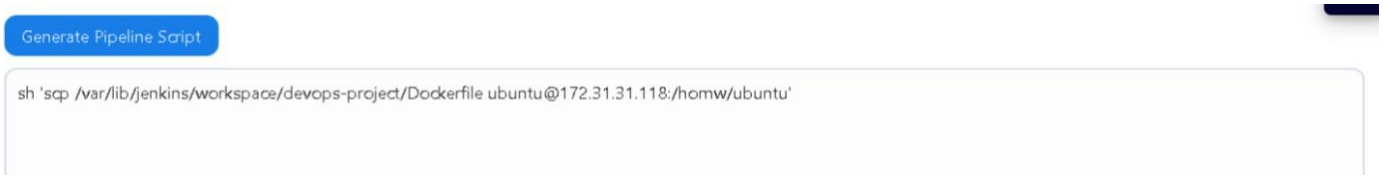
Next one is files creatation**: ansible.yml,demployment.yml,service.yml**



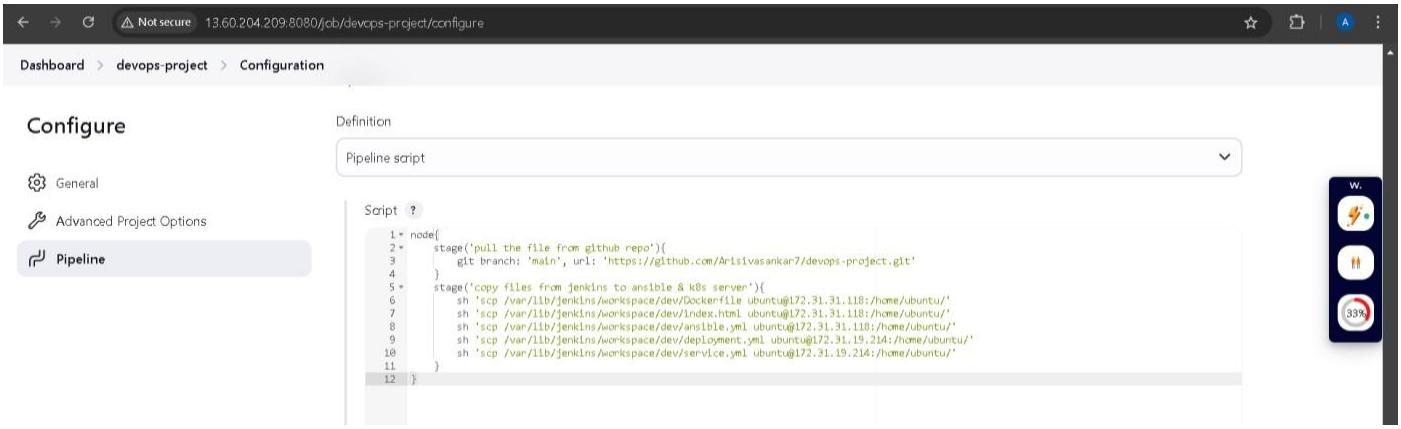
Then automatically transfer files in server because already created a github webhook so push that files github then Jenkins job automatically trigger.

Then go back Jenkins page and click pipeline syntax and choose **sh: shell script**



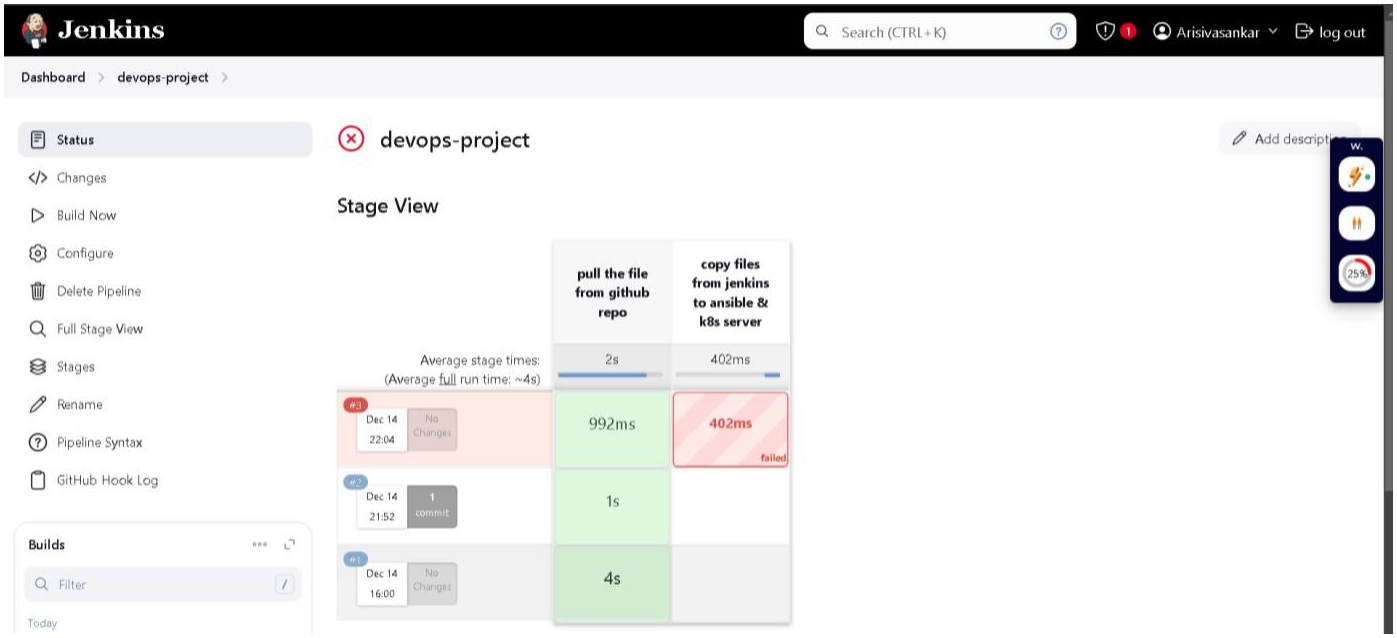


Then copy this command and paste it stage script:

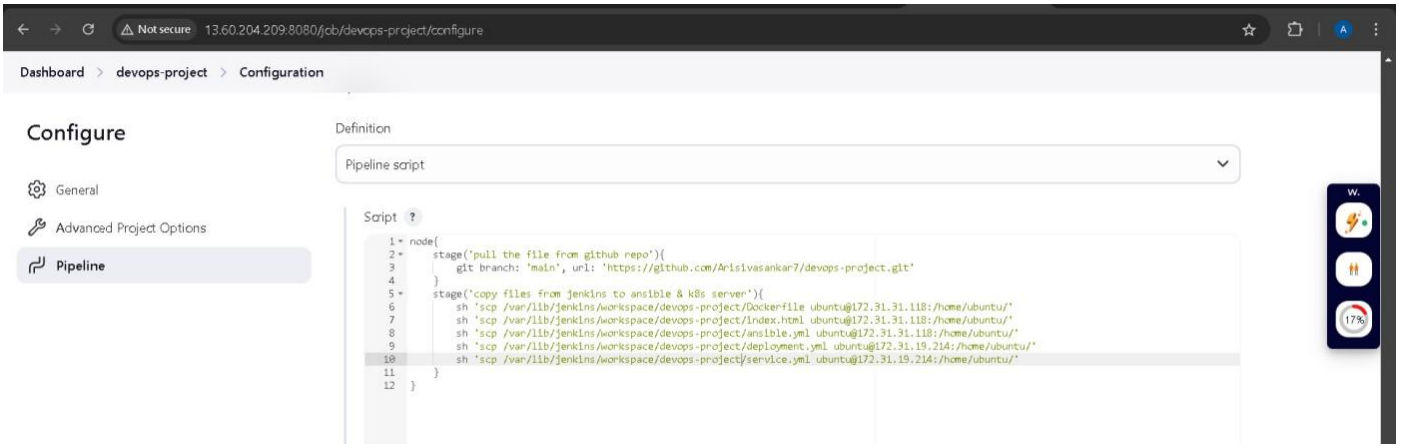


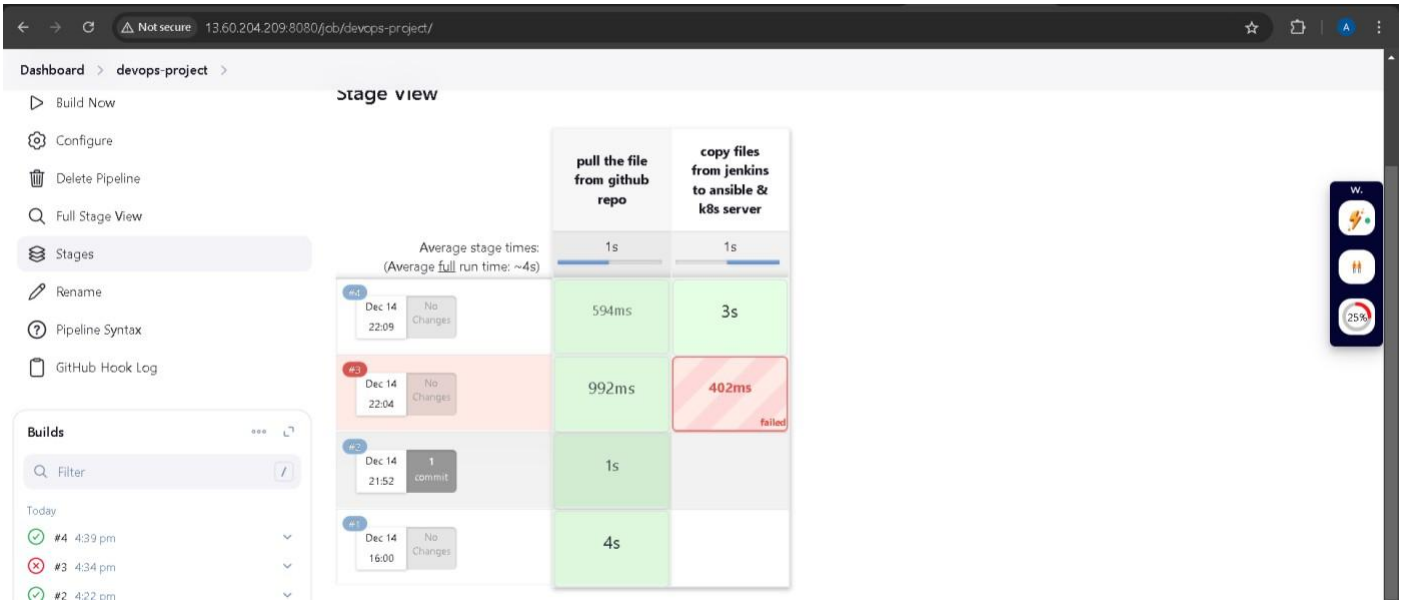
Then I was face one error because type the wrong job name that' why error then change my job name

Pipeline stage view:



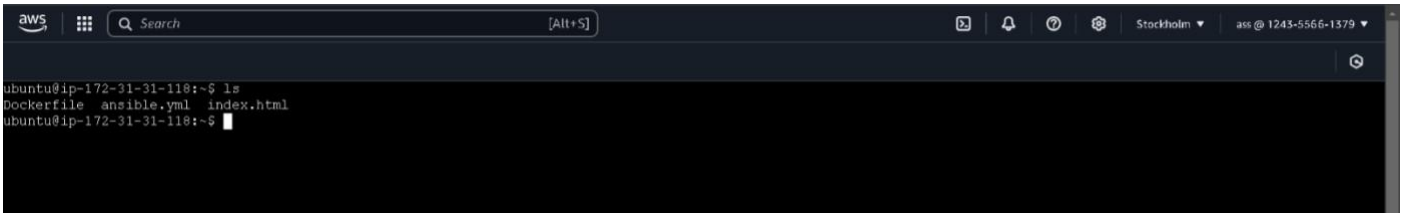
Again recreate script then click apply and save next run job





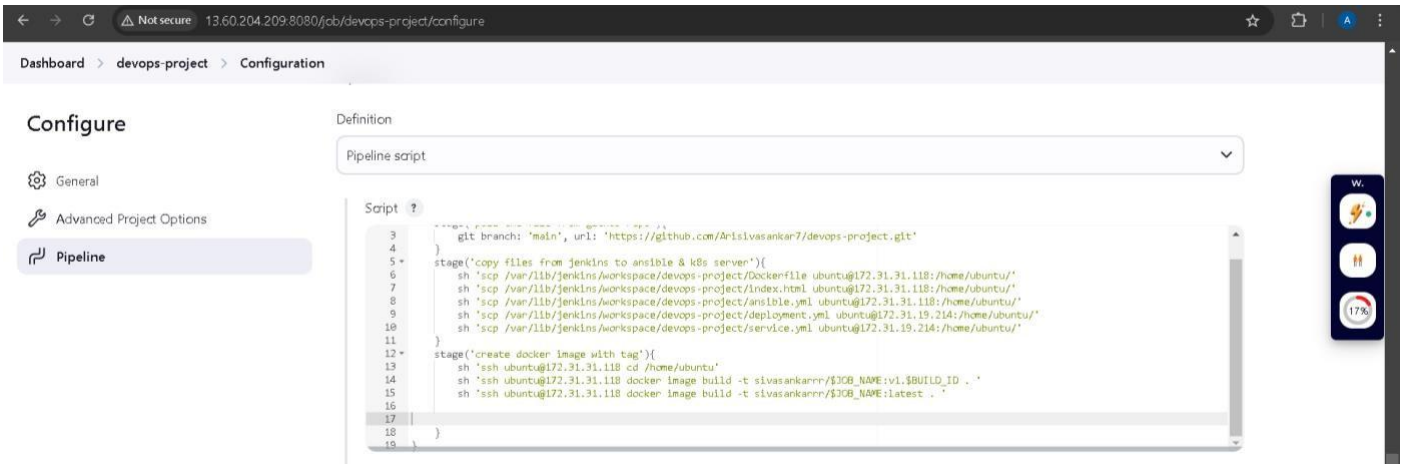
Then check my Ansible server for those files that are transferred.

Ansible server:

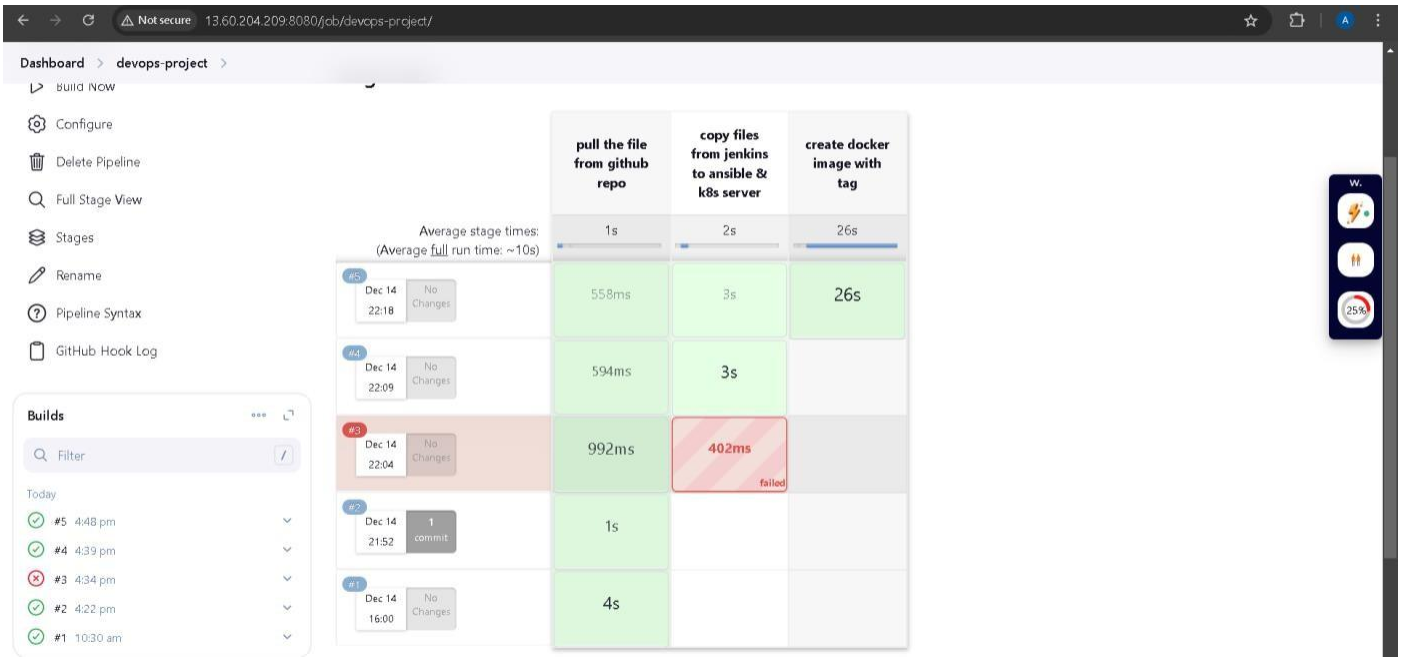


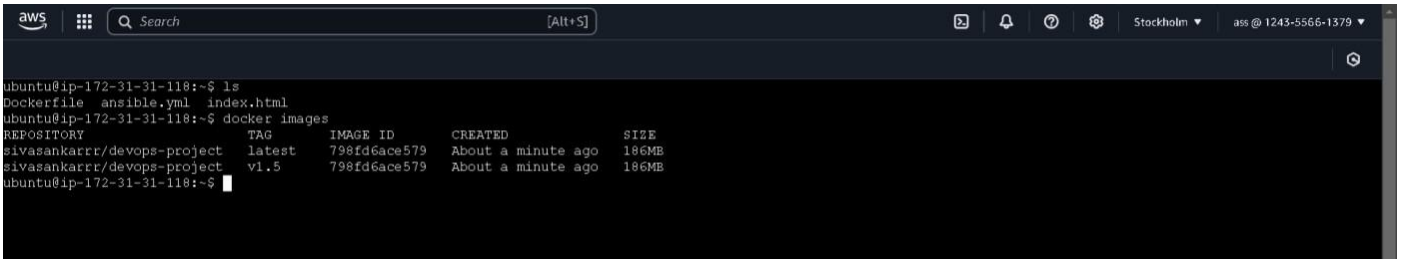
Master server:

Create docker image with tag script:



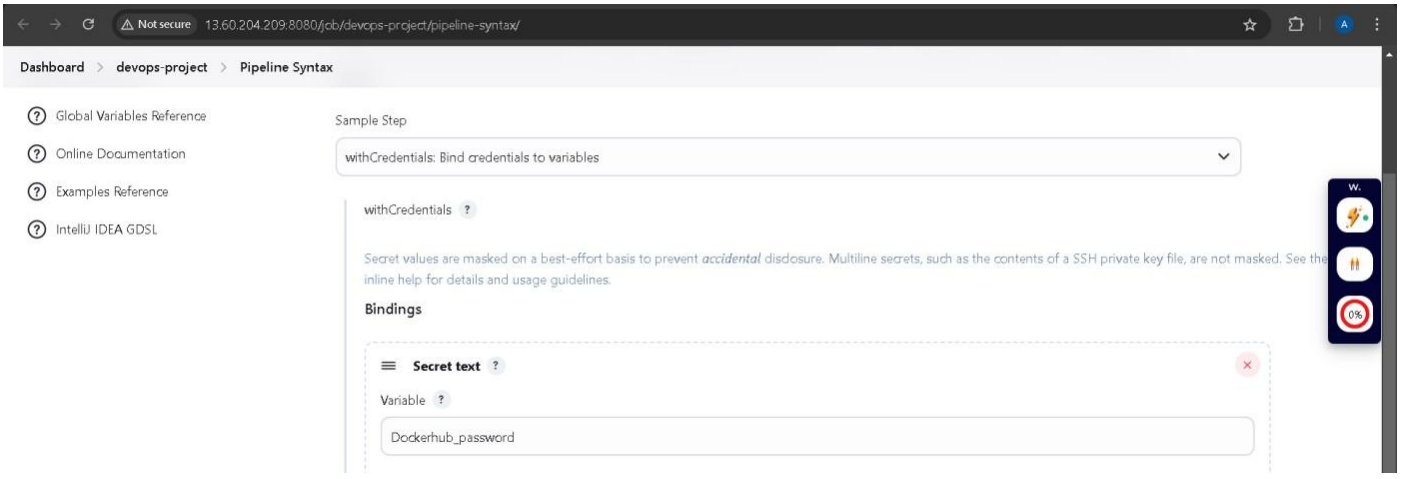
Successfully run those scripts.





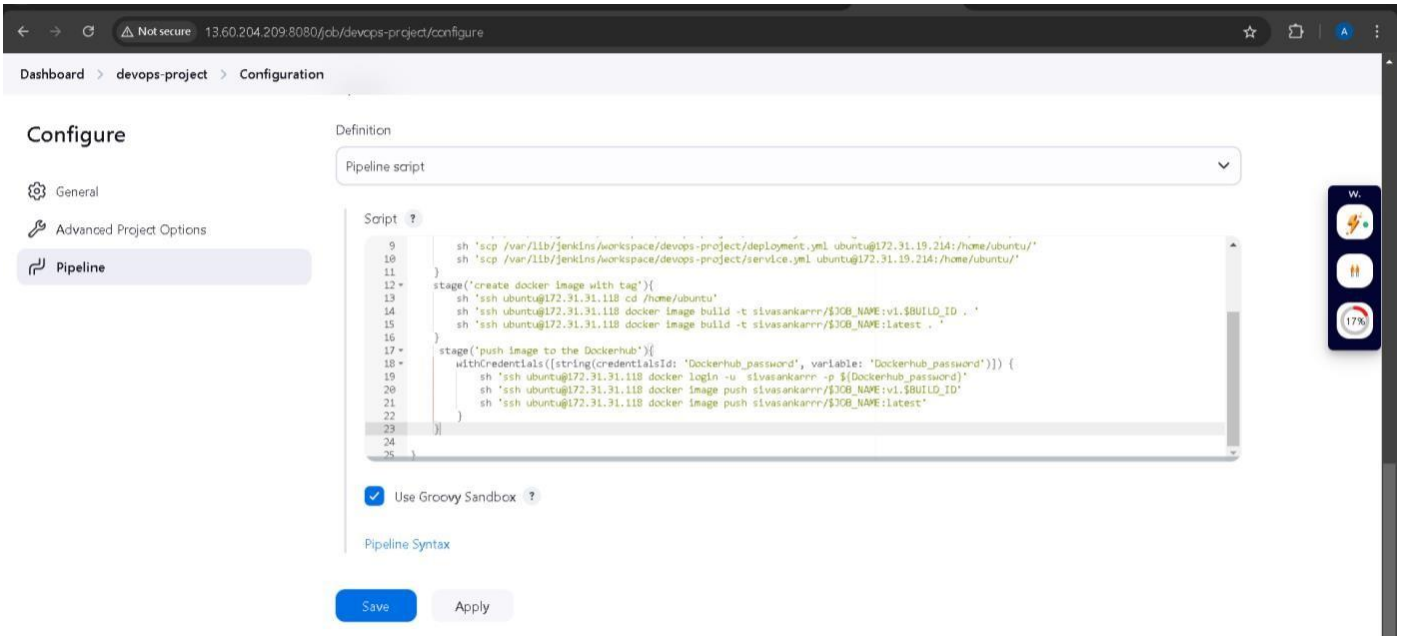
The next step is to push image to docker hub.

Some configura do that dockerhub password before push image to dockerhub. just because update raw dockerhub password in script it not safe that's why I do that some credentials for dockerhub



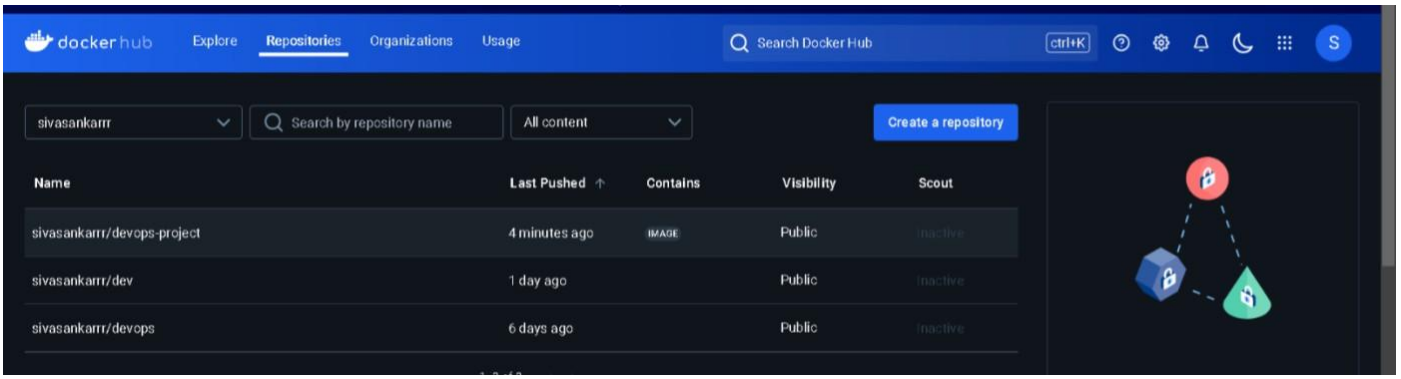
Next step is give the dockerhub password then create a id name and then click the apply button.

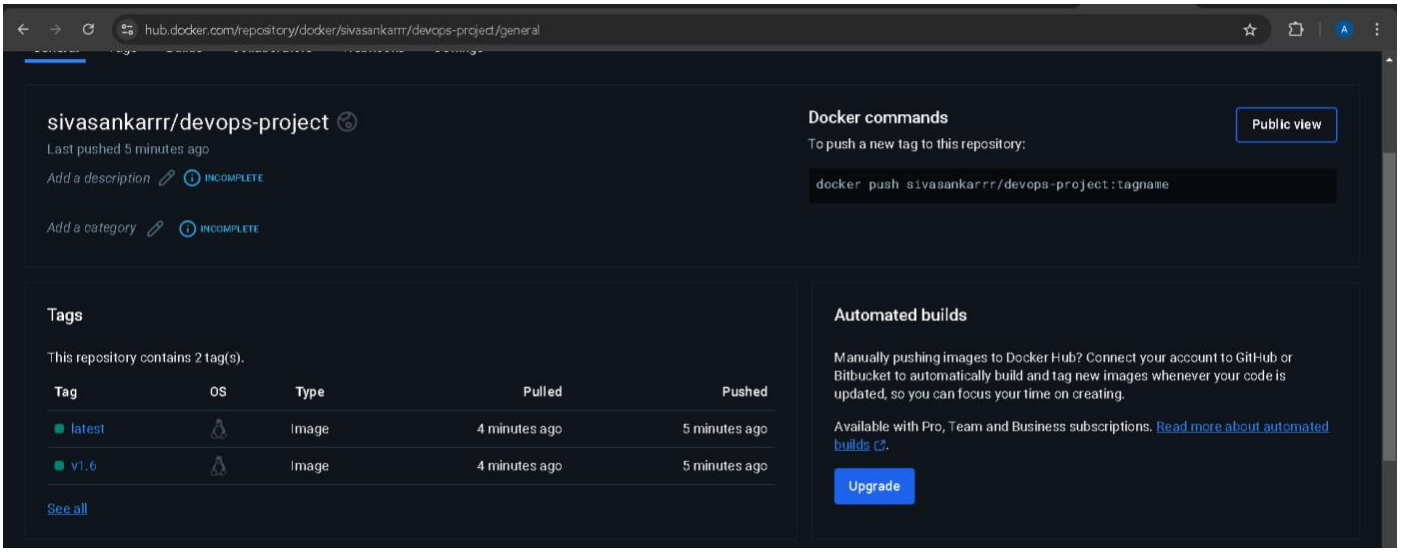
Go back stage script page. Create a script for push image to dockerhub and dockerhub login in ansible server before click apply.



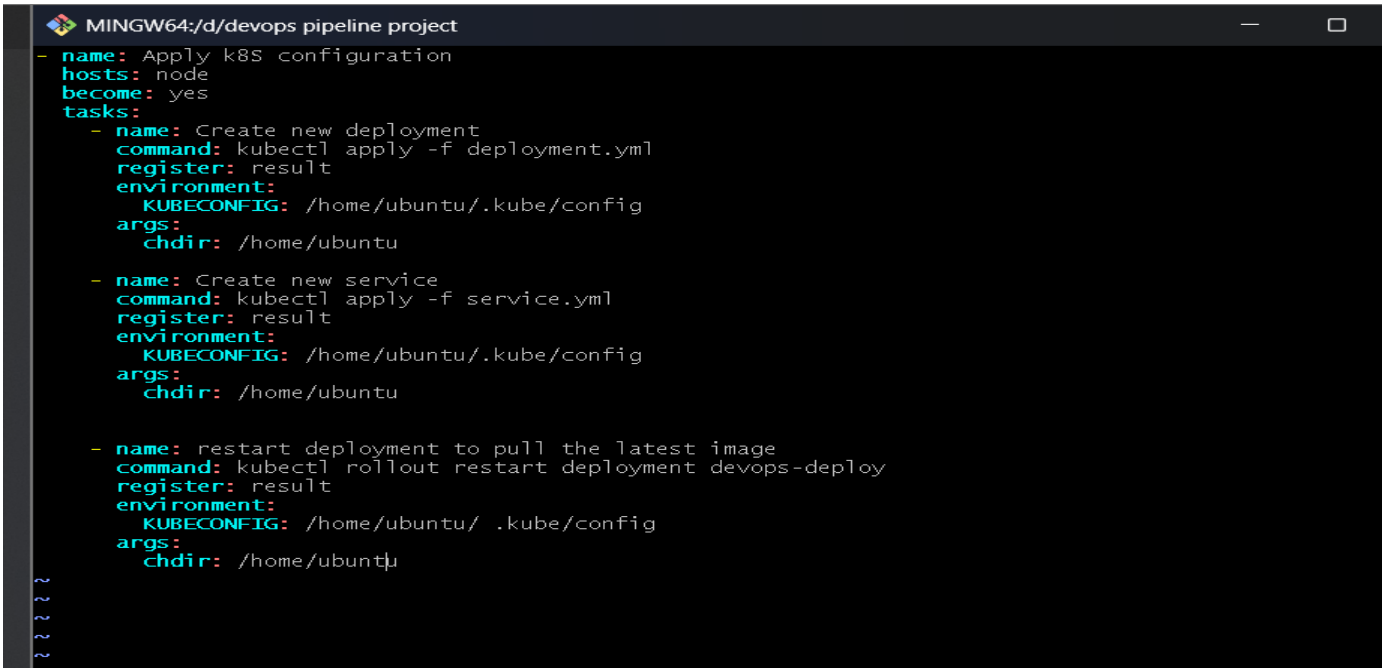
Then click the build now.

This is my dockerhub account and it’s successfully push image to my dockerhub.





The next step is write the yaml script for three yaml file. The first yml file use for run both yaml file I mean the first yaml file work by ansible playbook.



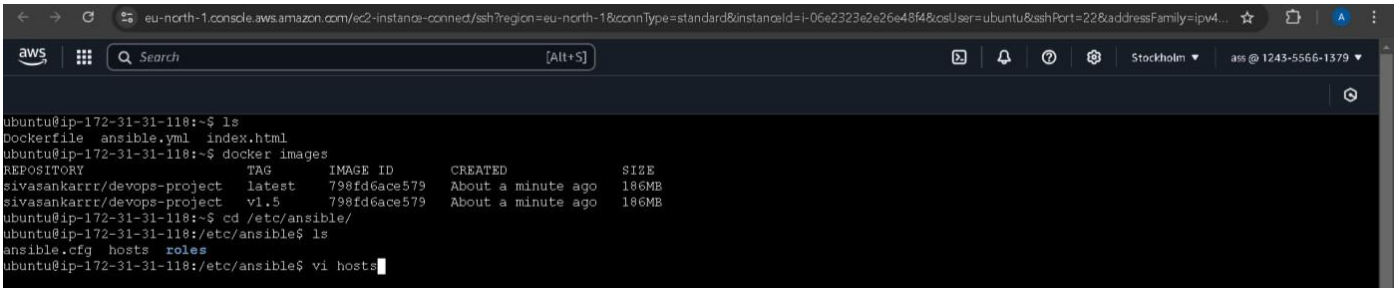
Deployment.yaml:

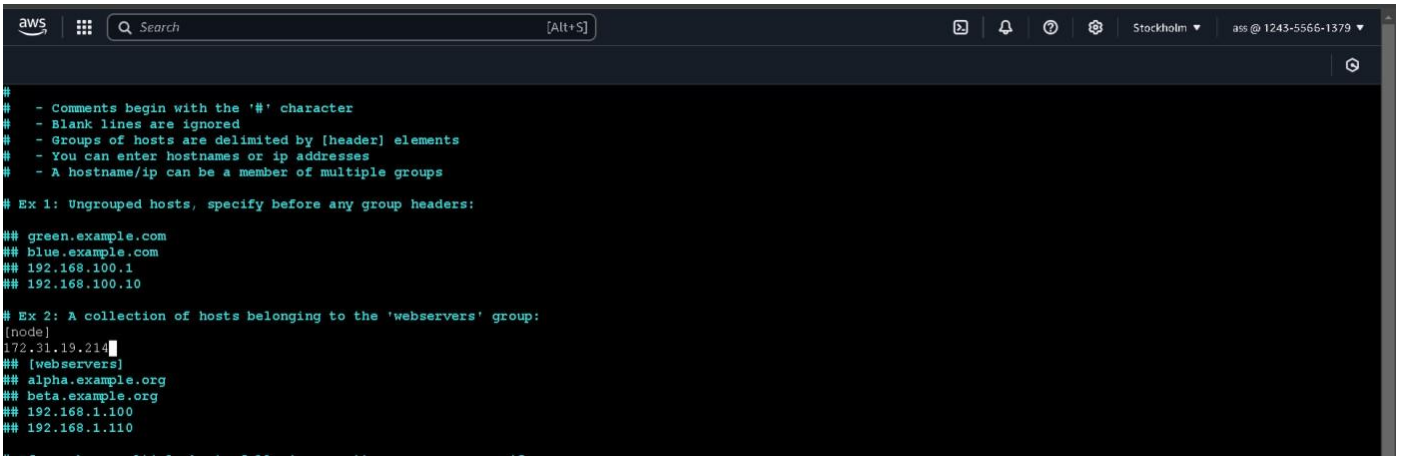
MINGW64:/d/devops pipeline project

```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: devops
spec:
  replicas: 3
  selector:
    matchLabels:
      app: devops
  template:
    metadata:
      labels:
        app: devops
    spec:
      containers:
      - name: devops
        image: sivasankarrr/dev
        imagePullPolicy: Always
        ports:
        - containerPort: 80

~
```

Service.yaml:

Push those all yaml file to server after it change some configura within ansible server.





The configuration finish those file and then it ‘s create ssh key and copy master server private ip address within ansible server because if it don’t connect to each server then it’s never properly run that yaml file that’s why it is must be copy those private ip .in ansible server.

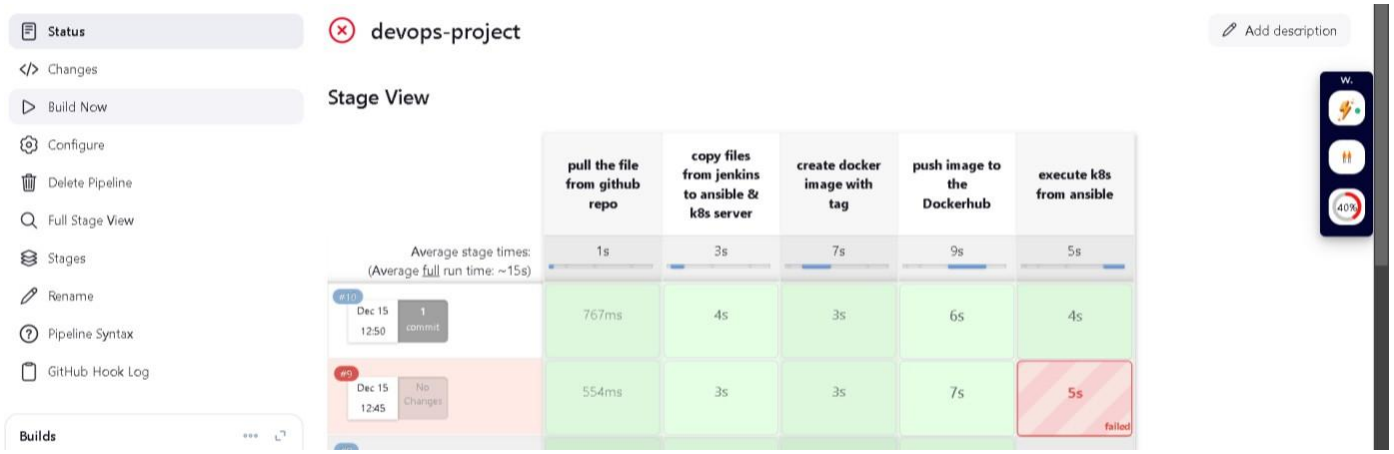
Push all three yaml file to github and then automatically trigger Jenkins script. After those edited three file store in ansible server and master machine.

```
arisi@arisivasankar MINGW64 /d/devops pipeline project (main)
$ vi ansible.yml

arisi@arisivasankar MINGW64 /d/devops pipeline project (main)
$ vi ansible.yml

arisi@arisivasankar MINGW64 /d/devops pipeline project (main)
$ vi deployment.yml

arisi@arisivasankar MINGW64 /d/devops pipeline project (main)
$ vi service.yml

arisi@arisivasankar MINGW64 /d/devops pipeline project (main)
$ vi service.yml

arisi@arisivasankar MINGW64 /d/devops pipeline project (main)
$ git add .
warning: in the working copy of 'ansible.yml', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'deployment.yml', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'service.yml', LF will be replaced by CRLF the next time Git touches it

arisi@arisivasankar MINGW64 /d/devops pipeline project (main)
$ git commit -m "update files"
[main 26ca71c] update files
 3 files changed, 67 insertions(+)

arisi@arisivasankar MINGW64 /d/devops pipeline project (main)
$ git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 957 bytes | 319.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Arisivasankar7/devops-project.git
   9340e81..26ca71c  main -> main

arisi@arisivasankar MINGW64 /d/devops pipeline project (main)
$
```

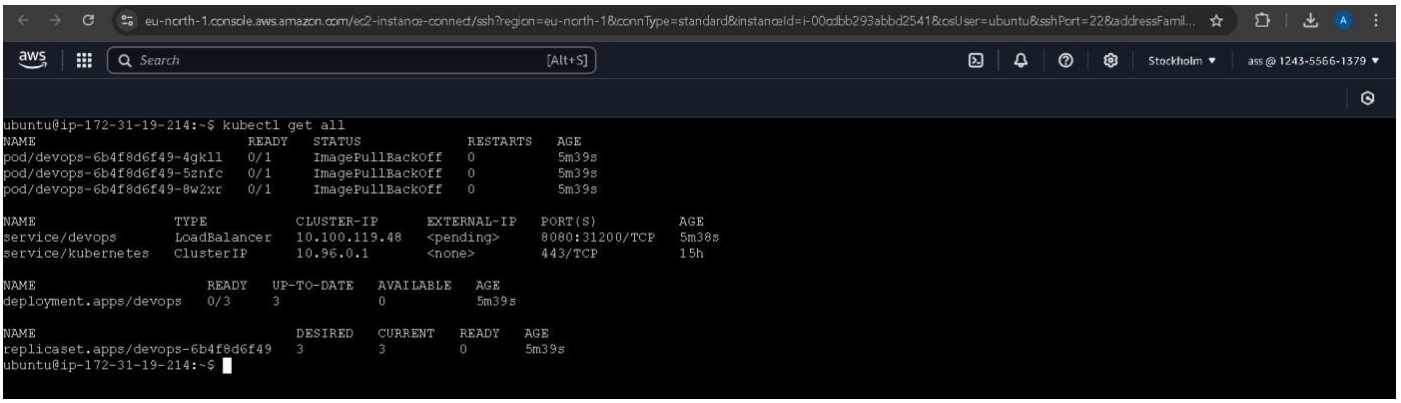
Successfully push all files.

Finally run the last stage script. This script do run the ansible playbook ansible.yml file

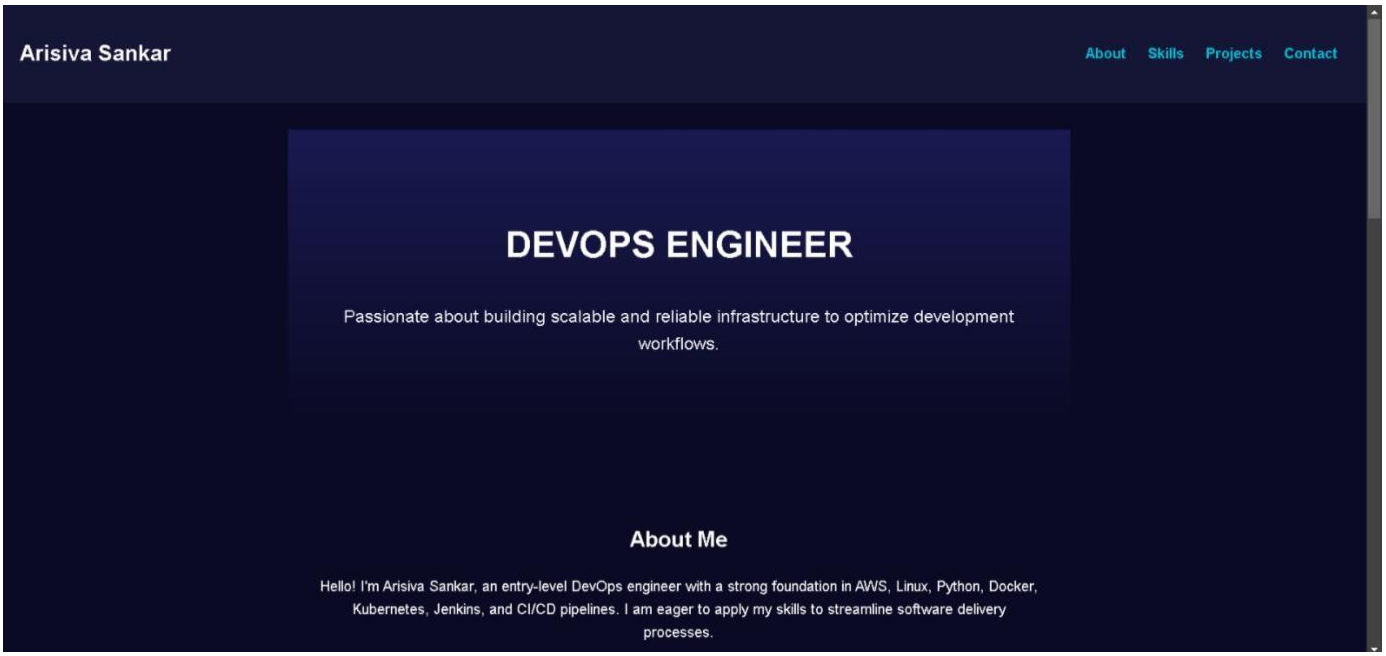
```
stage('execute k8s from ansible'){
        sh 'ssh ubuntu@172.31.28.53 cd /home/ubuntu'
        sh 'ssh ubuntu@172.31.28.53 ansible-playbook ansible.yml'
    }
```



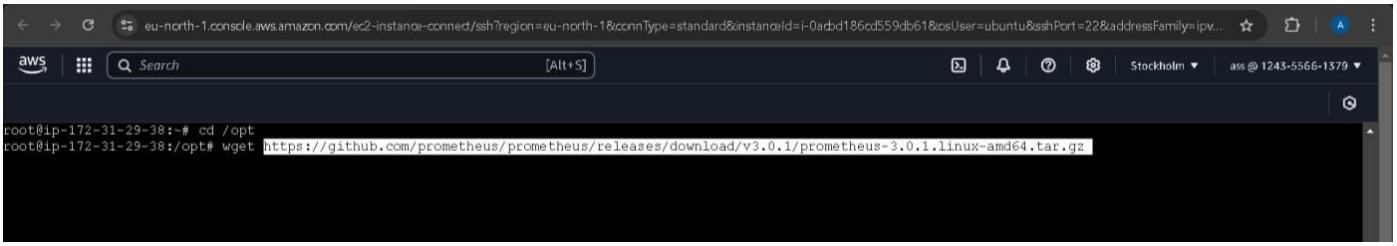
Finally job is successfully run then go back the master server. after type this command **(kubectl get all)**



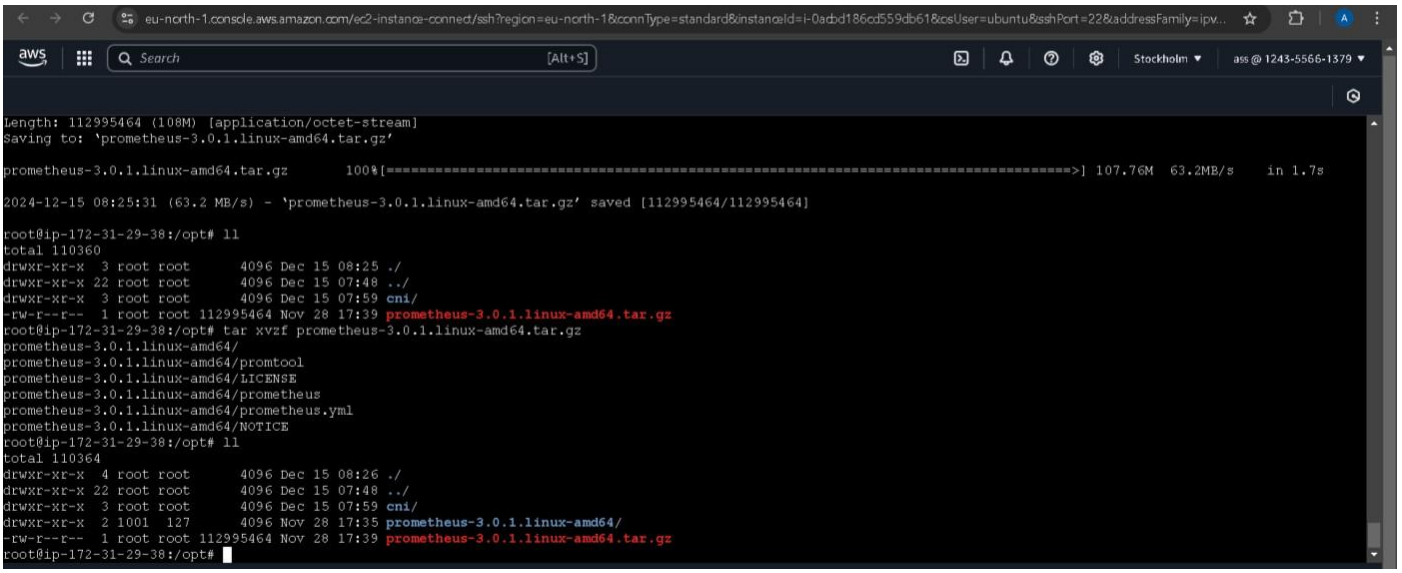
Then finally check the project output through browser.



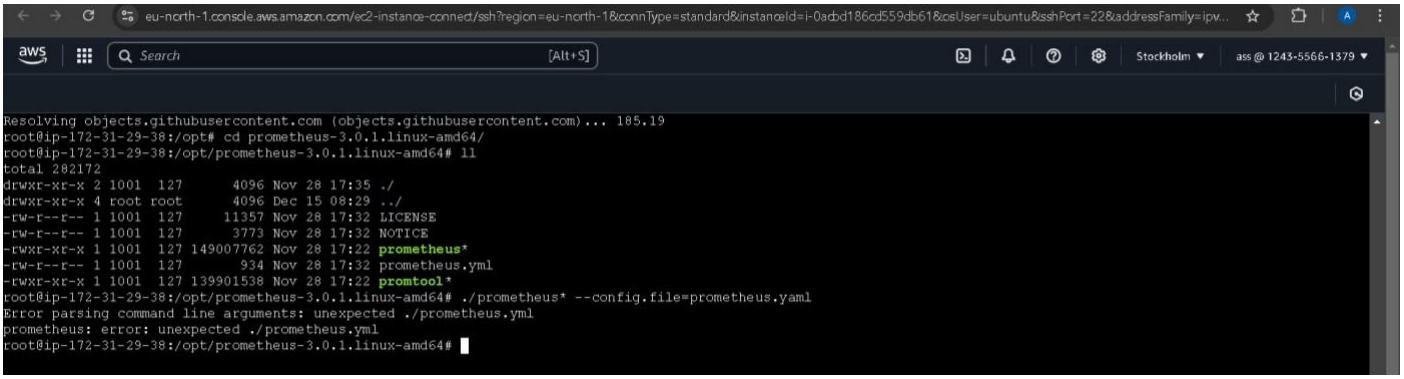
The next one is Prometheus installation. The Prometheus use for metrics monitoring purpose and it helpful tool because sometimes we don't know how many memory we are using for server, network traffic, and mostly use this tool for cpu utilization because we don't down cpu contition then it come biggest problem because we don't know which time server down that' why it's important.

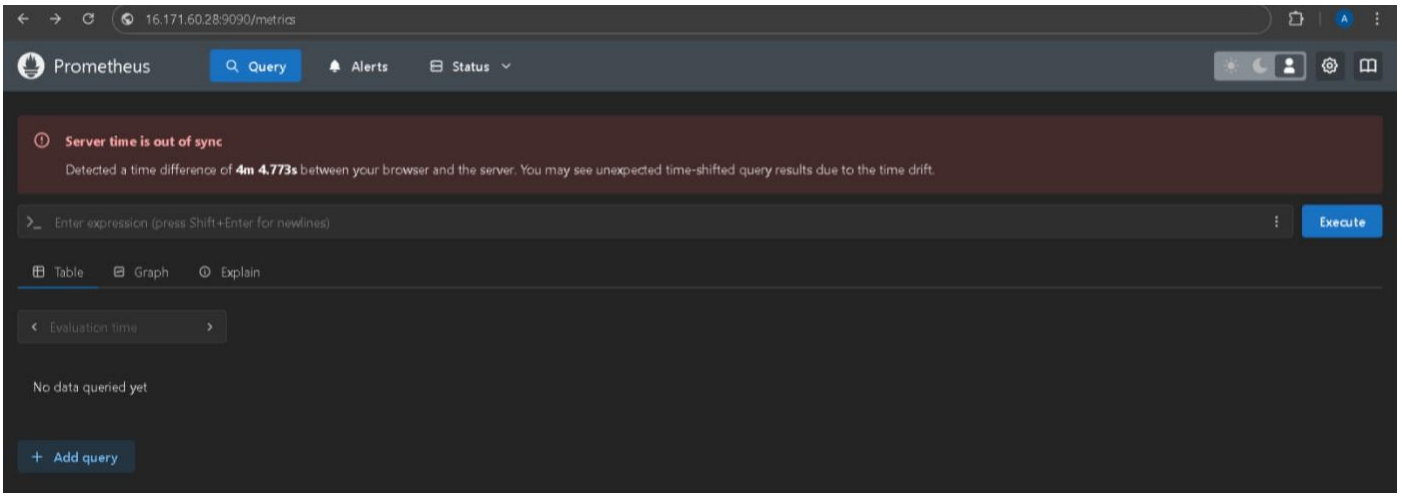


Successfully install that Prometheus tool in server then xtrack Prometheus file next run the Prometheus.

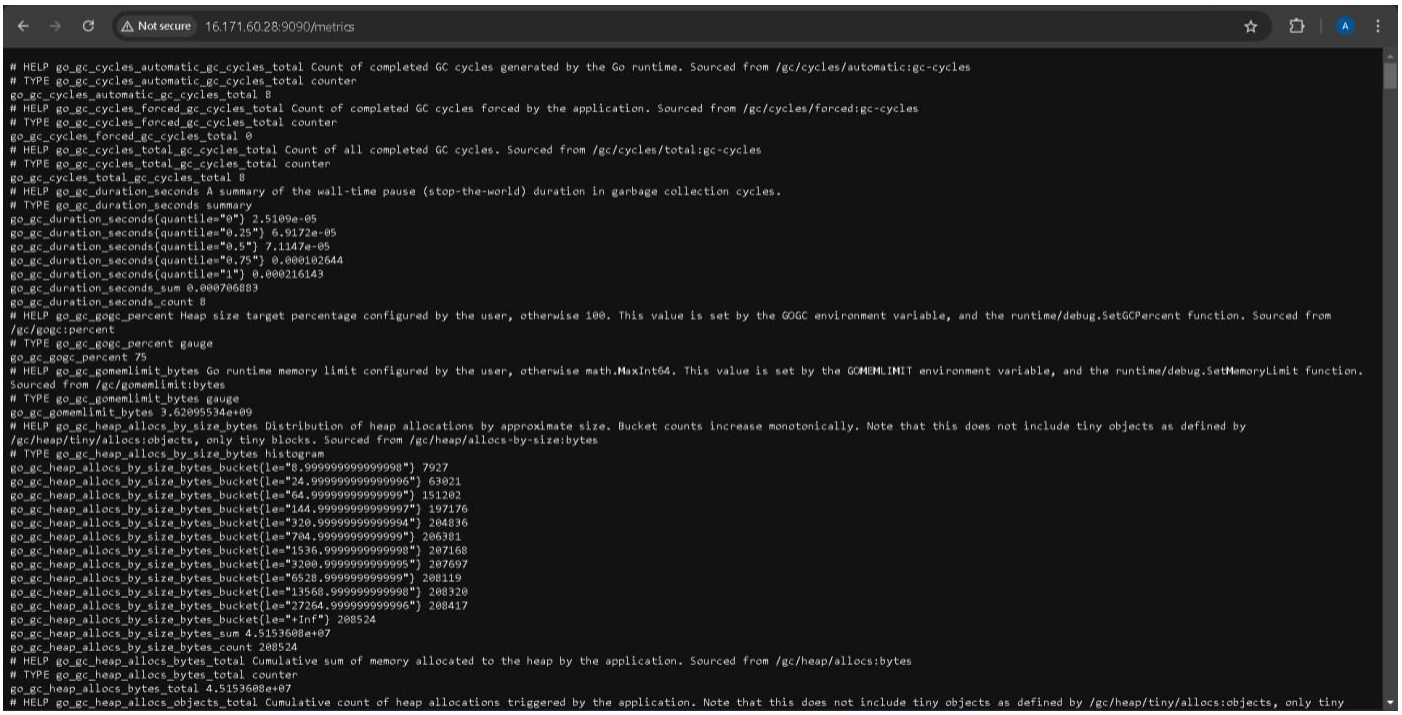


Run the Prometheus.



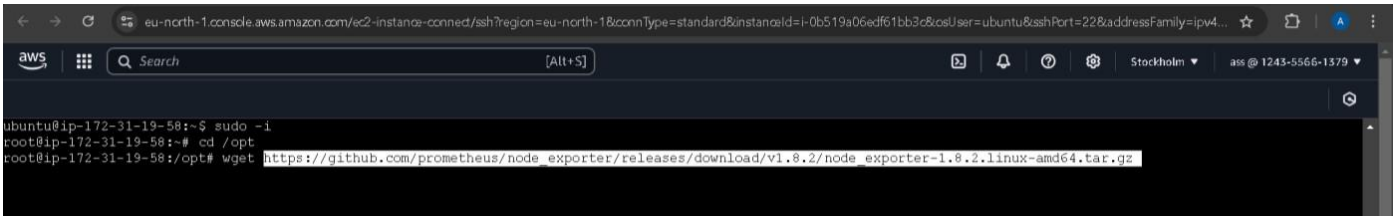


This is the metrics.



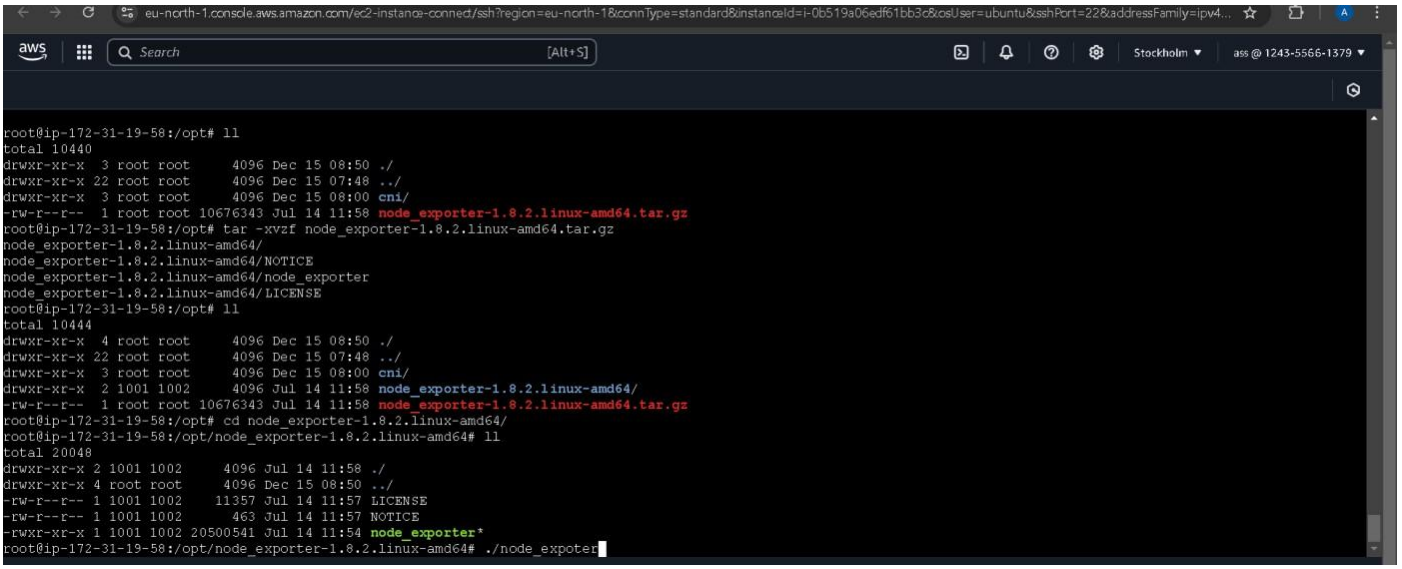

```
# HELP go_gc_cycles_automatic_gc_cycles_total Count of completed GC cycles generated by the Go runtime. Sourced from /gc/cycles/automatic:gc-cycles
# TYPE go_gc_cycles_automatic_gc_cycles_total counter
go_gc_cycles_automatic_gc_cycles_total 8
# HELP go_gc_cycles_forced_gc_cycles_total Count of completed GC cycles forced by the application. Sourced from /gc/cycles/forced:gc-cycles
# TYPE go_gc_cycles_forced_gc_cycles_total counter
go_gc_cycles_forced_gc_cycles_total 0
# HELP go_gc_cycles_total_gc_cycles_total Count of all completed GC cycles. Sourced from /gc/cycles/total:gc-cycles
# TYPE go_gc_cycles_total_gc_cycles_total counter
go_gc_cycles_total_gc_cycles_total 8
# HELP go_gc_duration_seconds A summary of the wall-time pause (stop-the-world) duration in garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 2.5109e-05
go_gc_duration_seconds{quantile="0.25"} 6.9172e-05
go_gc_duration_seconds{quantile="0.5"} 7.1147e-05
go_gc_duration_seconds{quantile="0.75"} 0.000102644
go_gc_duration_seconds{quantile="1"} 0.000216143
go_gc_duration_seconds_sum 0.000706883
go_gc_duration_seconds_count 8
# HELP go_gc_gogc_percent Heap size target percentage configured by the user, otherwise 100. This value is set by the GOGC environment variable, and the runtime/debug.SetGCPercent function. Sourced from
/gc/gogc:percent
# TYPE go_gc_gogc_percent gauge
go_gc_gogc_percent 75
# HELP go_gc_gomemlimit_bytes Go runtime memory limit configured by the user, otherwise math.MaxInt64. This value is set by the GOMEMLIMIT environment variable, and the runtime/debug.SetMemoryLimit function.
Sourced from /gc/gomemlimit:bytes
# TYPE go_gc_gomemlimit_bytes gauge
go_gc_gomemlimit_bytes 3.62095534e+09
# HELP go_gc_heap_allocs_by_size_bytes Distribution of heap allocations by approximate size. Bucket counts increase monotonically. Note that this does not include tiny objects as defined by
/gc/heap/tiny/allocs:objects, only tiny blocks. Sourced from /gc/heap/allocs-by-size:bytes
# TYPE go_gc_heap_allocs_by_size_bytes histogram
go_gc_heap_allocs_by_size_bytes_bucket{le="8.999999999999998"} 7927
go_gc_heap_allocs_by_size_bytes_bucket{le="24.999999999999996"} 63021
go_gc_heap_allocs_by_size_bytes_bucket{le="64.99999999999999"} 151202
go_gc_heap_allocs_by_size_bytes_bucket{le="144.99999999999997"} 197176
go_gc_heap_allocs_by_size_bytes_bucket{le="320.99999999999994"} 204836
go_gc_heap_allocs_by_size_bytes_bucket{le="704.9999999999999"} 206381
go_gc_heap_allocs_by_size_bytes_bucket{le="1536.9999999999998"} 207168
go_gc_heap_allocs_by_size_bytes_bucket{le="3200.9999999999995"} 207697
go_gc_heap_allocs_by_size_bytes_bucket{le="6528.999999999999"} 208119
go_gc_heap_allocs_by_size_bytes_bucket{le="13568.999999999998"} 208320
go_gc_heap_allocs_by_size_bytes_bucket{le="27264.999999999996"} 208417
go_gc_heap_allocs_by_size_bytes_bucket{le="+Inf"} 208524
go_gc_heap_allocs_by_size_bytes_sum 4.5153608e+07
go_gc_heap_allocs_by_size_bytes_count 208524
# HELP go_gc_heap_allocs_bytes_total Cumulative sum of memory allocated to the heap by the application. Sourced from /gc/heap/allocs:bytes
# TYPE go_gc_heap_allocs_bytes_total counter
go_gc_heap_allocs_bytes_total 4.5153608e+07
# HELP go_gc_heap_allocs_objects_total Cumulative count of heap allocations triggered by the application. Note that this does not include tiny objects as defined by /gc/heap/tiny/allocs:objects, only tiny
```

The next installation is the node_exporter. the node exporter work is collating the slave server metrics then push those slave metrics to master server. that is node exporter concept.



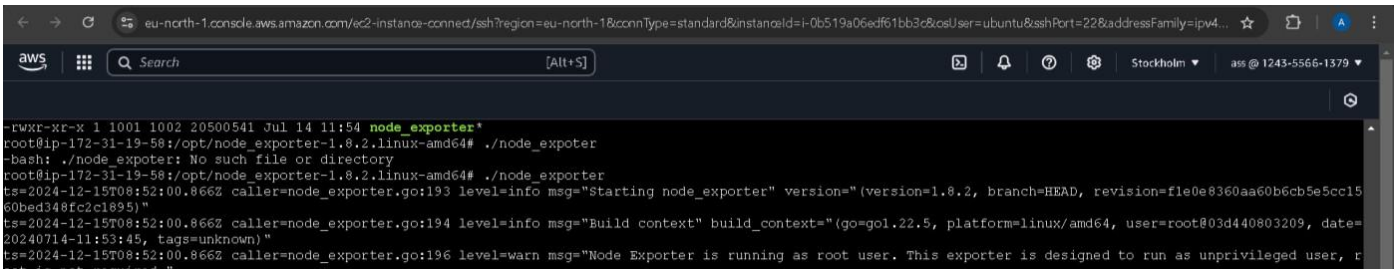

```
ubuntu@ip-172-31-19-58:~$ sudo -i
root@ip-172-31-19-58:~# cd /opt
root@ip-172-31-19-58:/opt# wget https://github.com/prometheus/node_exporter/releases/download/v1.8.2/node_exporter-1.8.2.linux-amd64.tar.gz
```

Then xtrack the node exporter file

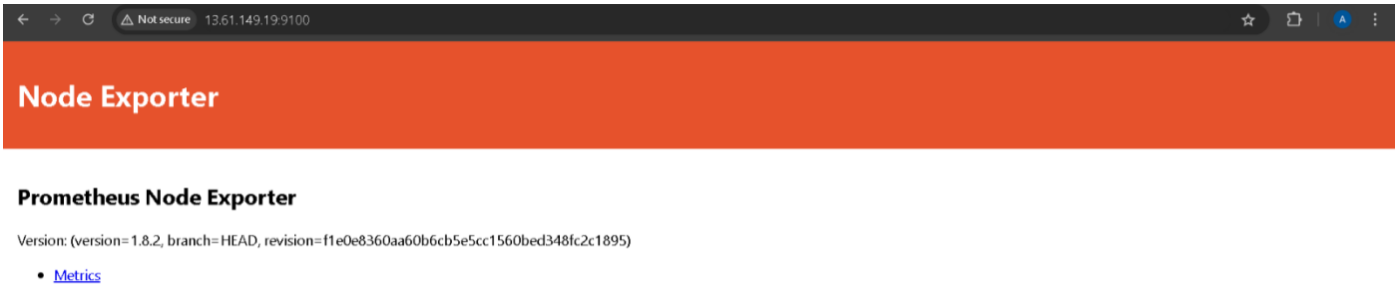



```
root@ip-172-31-19-58:/opt# ll
total 10440
drwxr-xr-x  3 root root     4096 Dec 15 08:50 ./
drwxr-xr-x 22 root root     4096 Dec 15 07:48 ../
drwxr-xr-x  3 root root     4096 Dec 15 08:00 cni/
-rw-r--r--  1 root root 10676343 Jul 14 11:58 node_exporter-1.8.2.linux-amd64.tar.gz
root@ip-172-31-19-58:/opt# tar -xvzf node_exporter-1.8.2.linux-amd64.tar.gz
node_exporter-1.8.2.linux-amd64/
node_exporter-1.8.2.linux-amd64/NOTICE
node_exporter-1.8.2.linux-amd64/node_exporter
node_exporter-1.8.2.linux-amd64/LICENSE
root@ip-172-31-19-58:/opt# ll
total 10444
drwxr-xr-x  4 root root     4096 Dec 15 08:50 ./
drwxr-xr-x 22 root root     4096 Dec 15 07:48 ../
drwxr-xr-x  3 root root     4096 Dec 15 08:00 cni/
drwxr-xr-x  2 1001 1002     4096 Jul 14 11:58 node_exporter-1.8.2.linux-amd64/
-rw-r--r--  1 root root 10676343 Jul 14 11:58 node_exporter-1.8.2.linux-amd64.tar.gz
root@ip-172-31-19-58:/opt# cd node_exporter-1.8.2.linux-amd64/
root@ip-172-31-19-58:/opt/node_exporter-1.8.2.linux-amd64# ll
total 20048
drwxr-xr-x 2 1001 1002     4096 Jul 14 11:58 ./
drwxr-xr-x 4 root root     4096 Dec 15 08:50 ../
-rw-r--r-- 1 1001 1002    11357 Jul 14 11:57 LICENSE
-rw-r--r-- 1 1001 1002      463 Jul 14 11:57 NOTICE
-rwxr-xr-x 1 1001 1002 20500541 Jul 14 11:54 node_exporter*
root@ip-172-31-19-58:/opt/node_exporter-1.8.2.linux-amd64# ./node_expoter
```

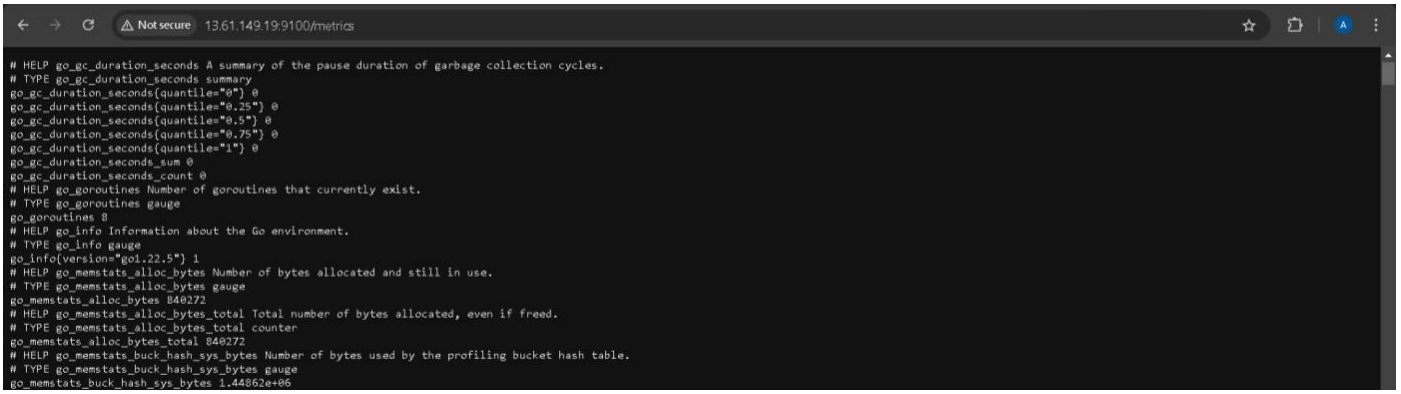
Finally run the node exporter.



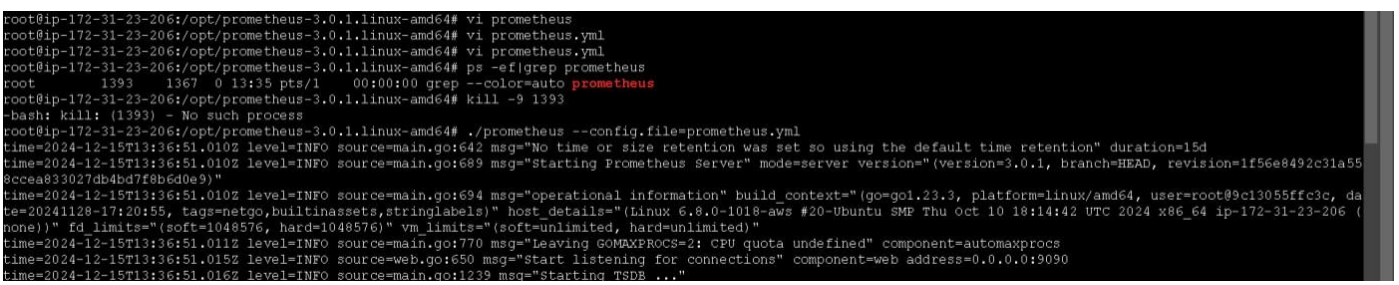
Node exporter home page.



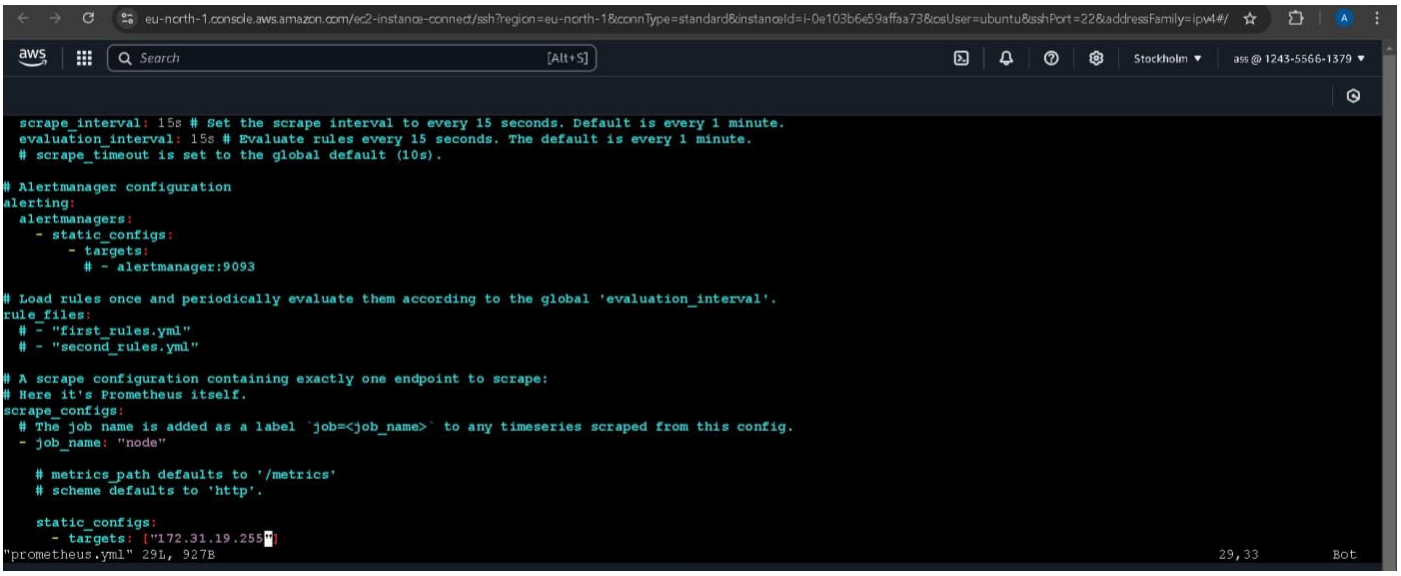
Slave server metrics.



The next process is the connected the master server Prometheus to slave server node exporter.



Set job name and targets:



:wq!

Next type this command (ps -ef| grep Prometheus)

Kill command use for process kill

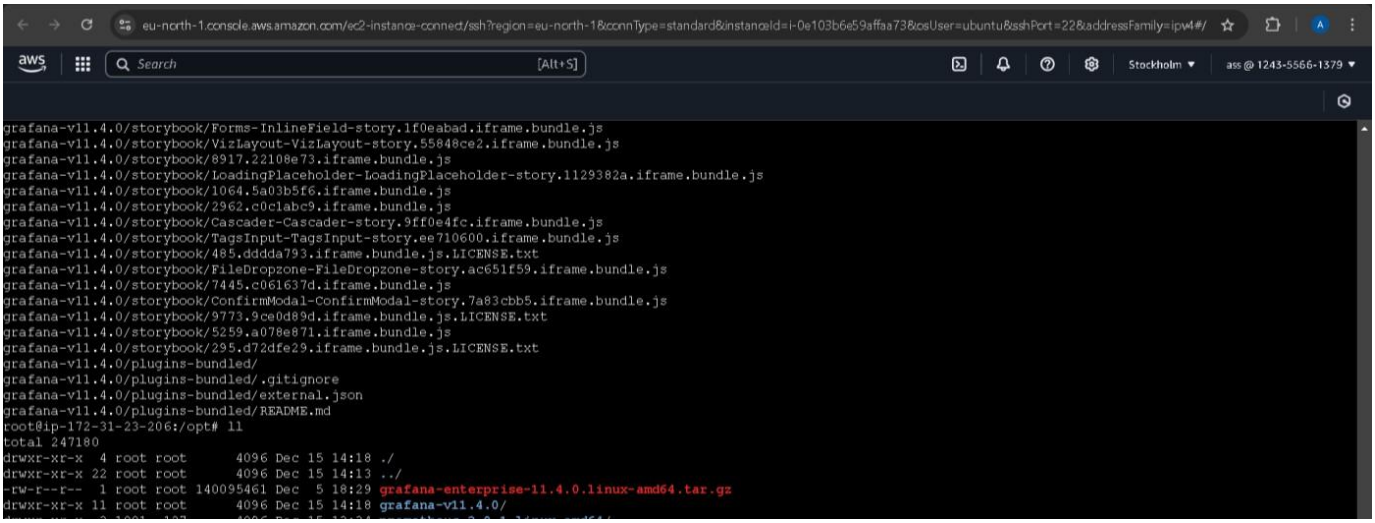
Again one more time run Prometheus.

./promrtheus –config.file=Prometheus.yml

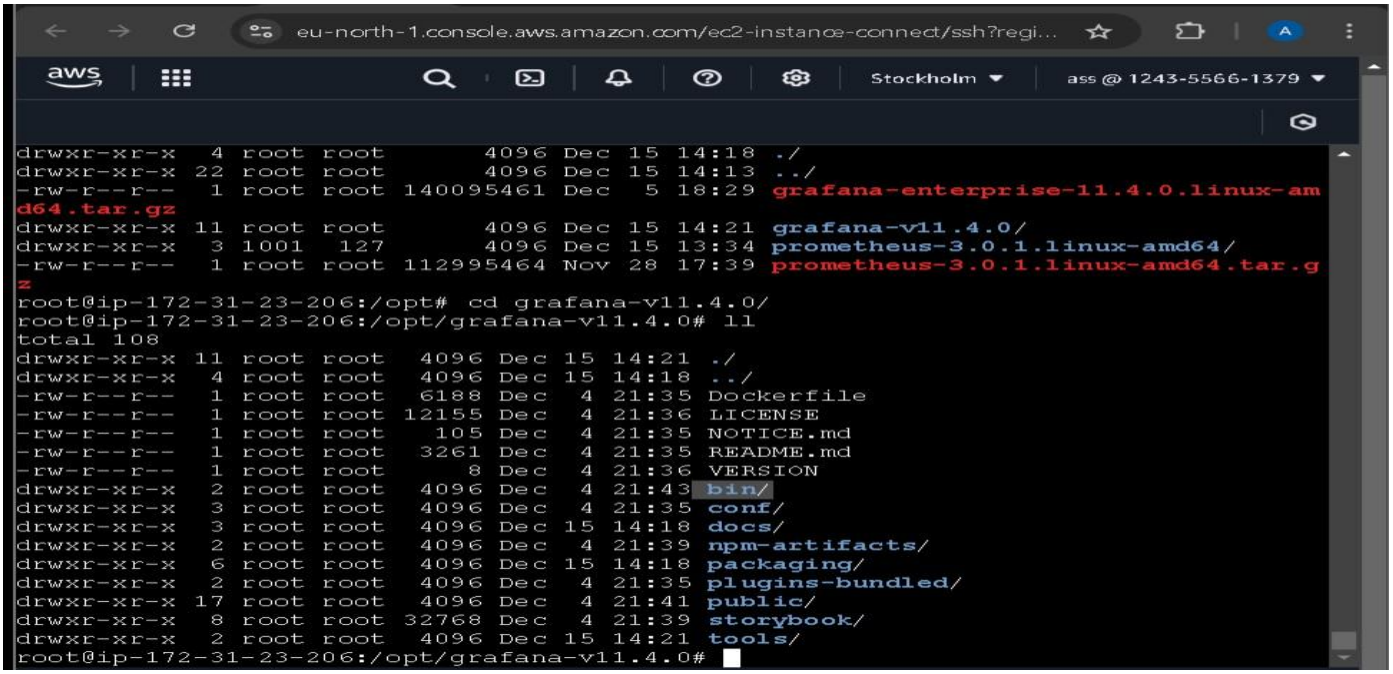
Finally successfully connected Prometheus and node exporter together.

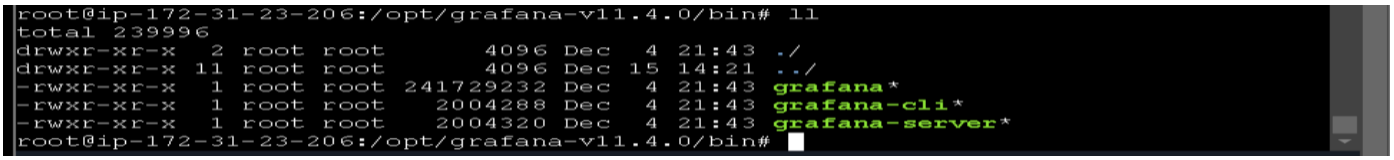
Then check the slave server metrics by master server because it already connected the booth tools

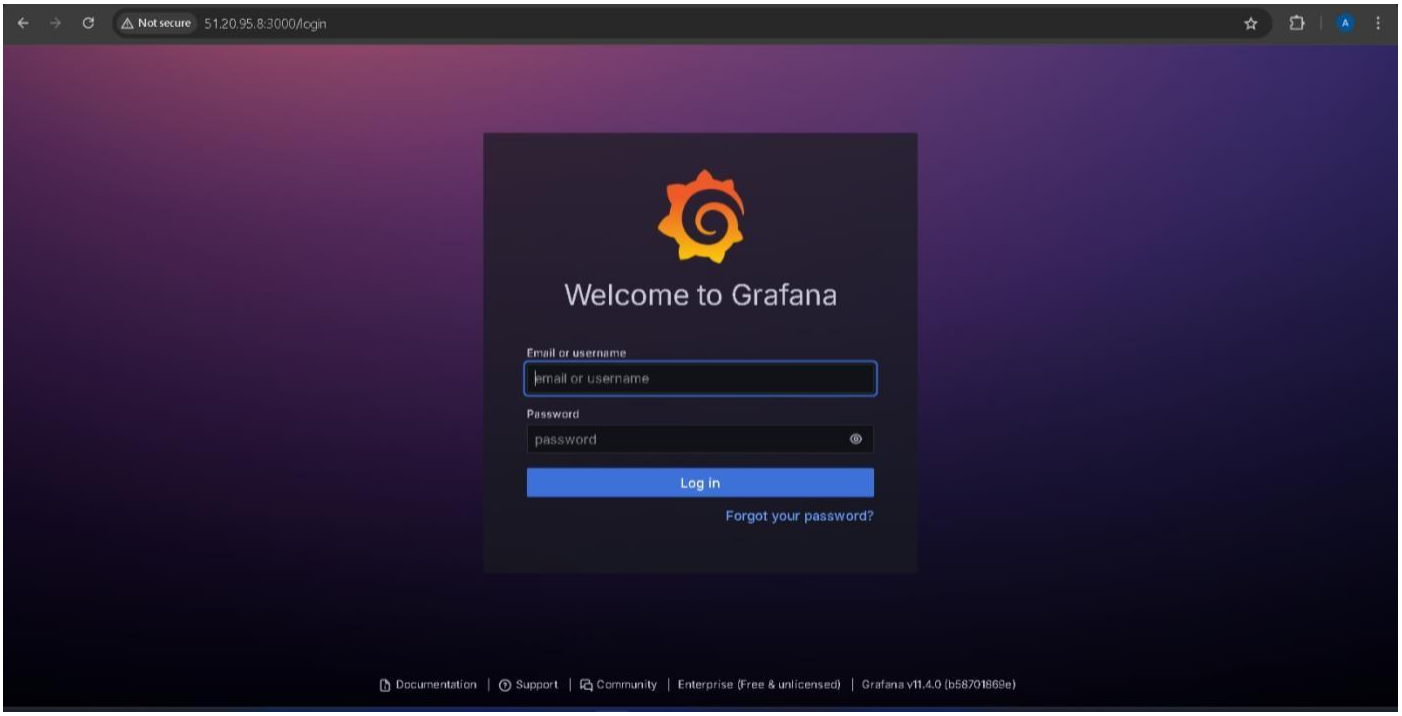
Finally install is Grafana tool.



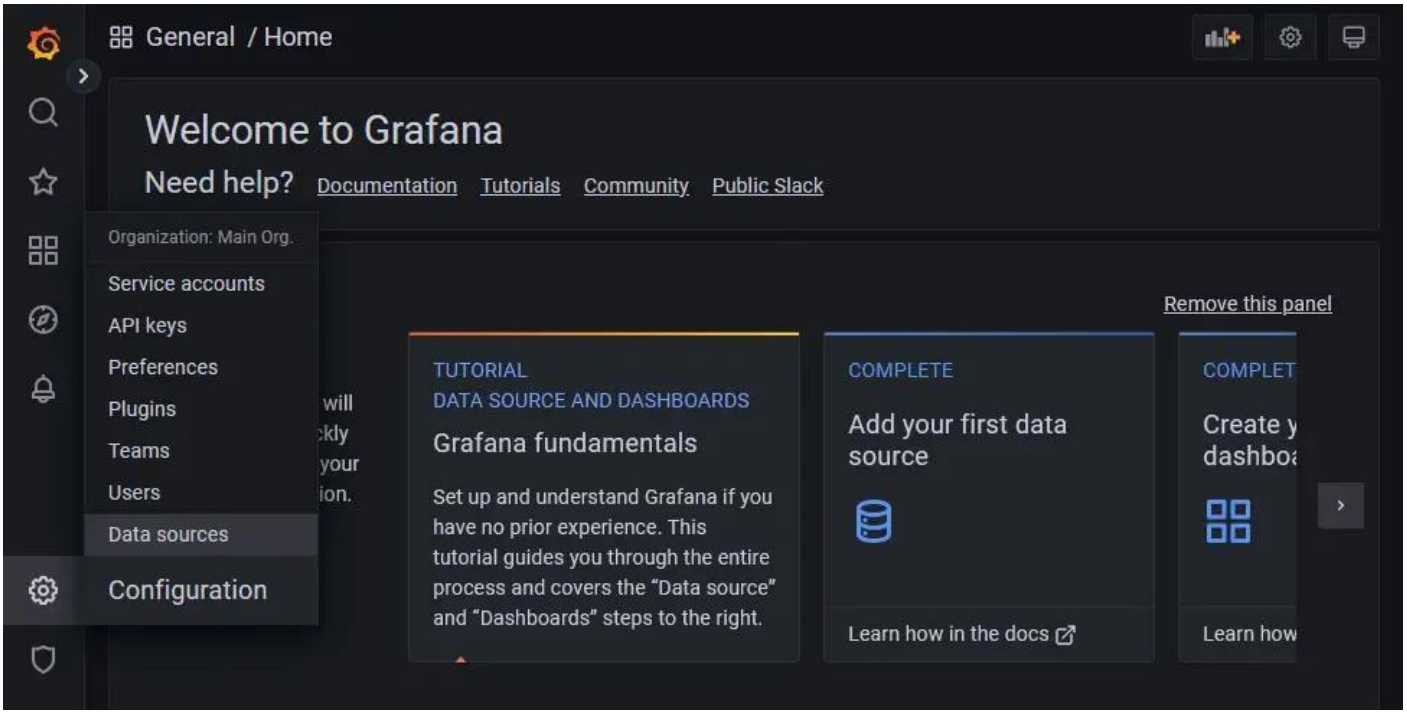
Xtrack Grafana file then gone bin path



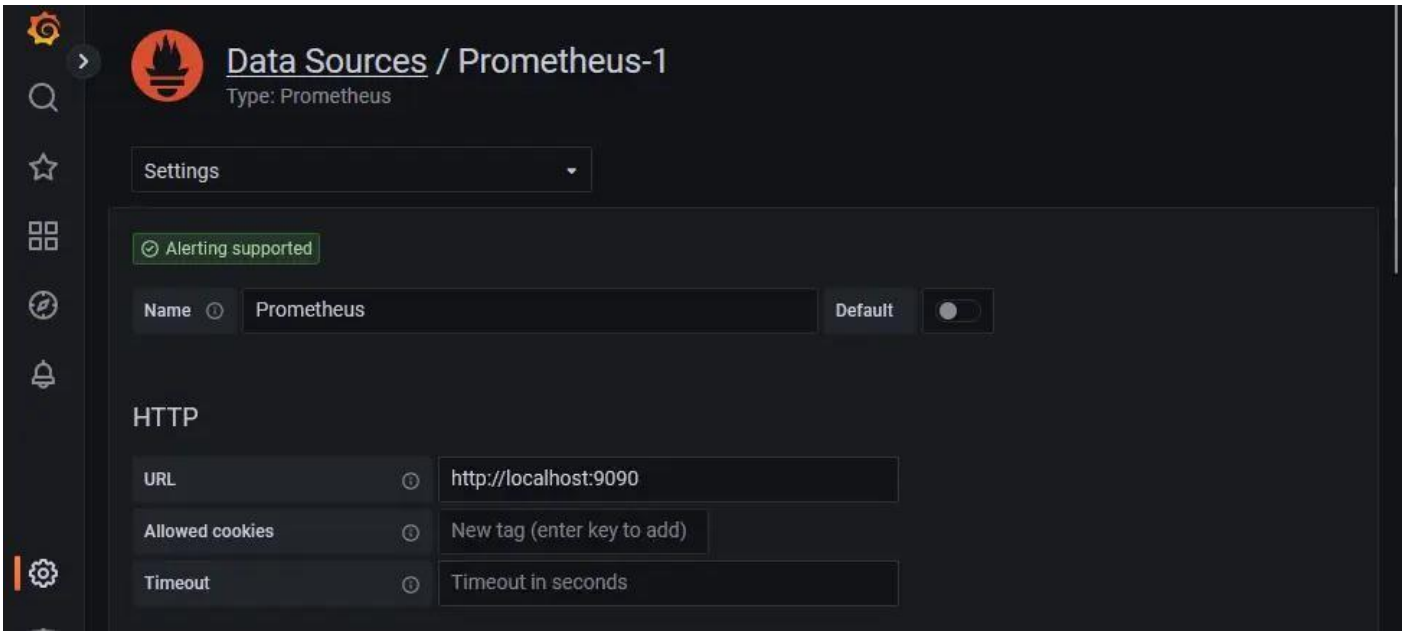


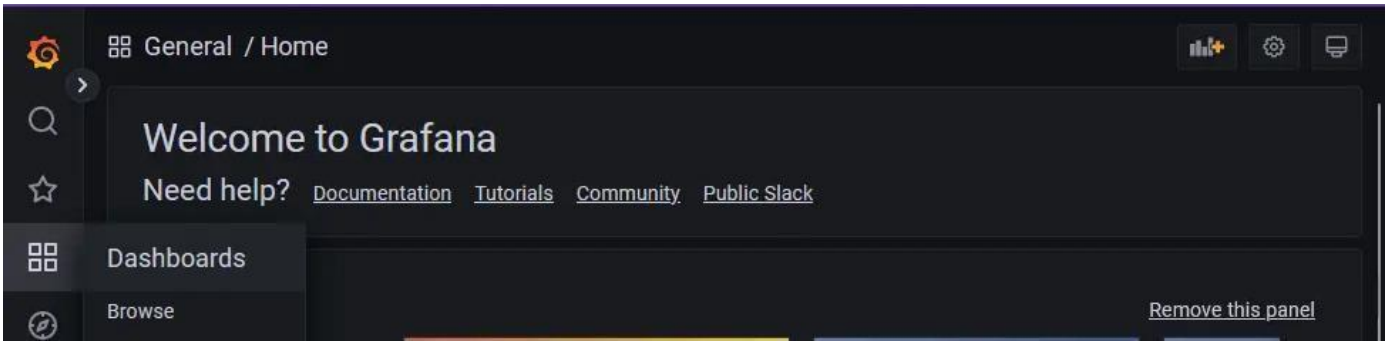


Checking the Prometheus Datasource:



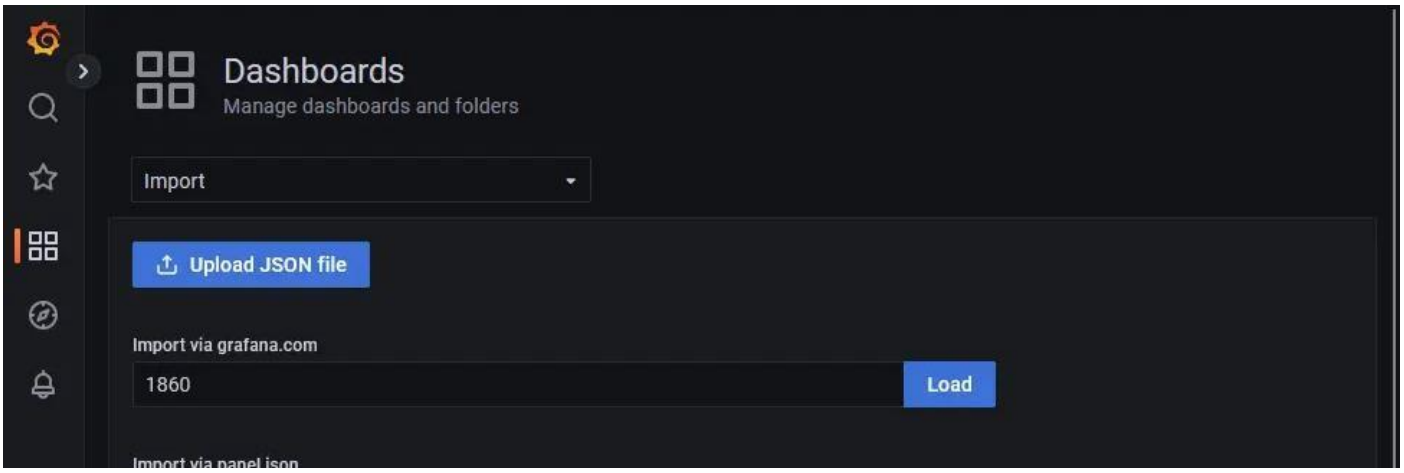
Select the datasources.



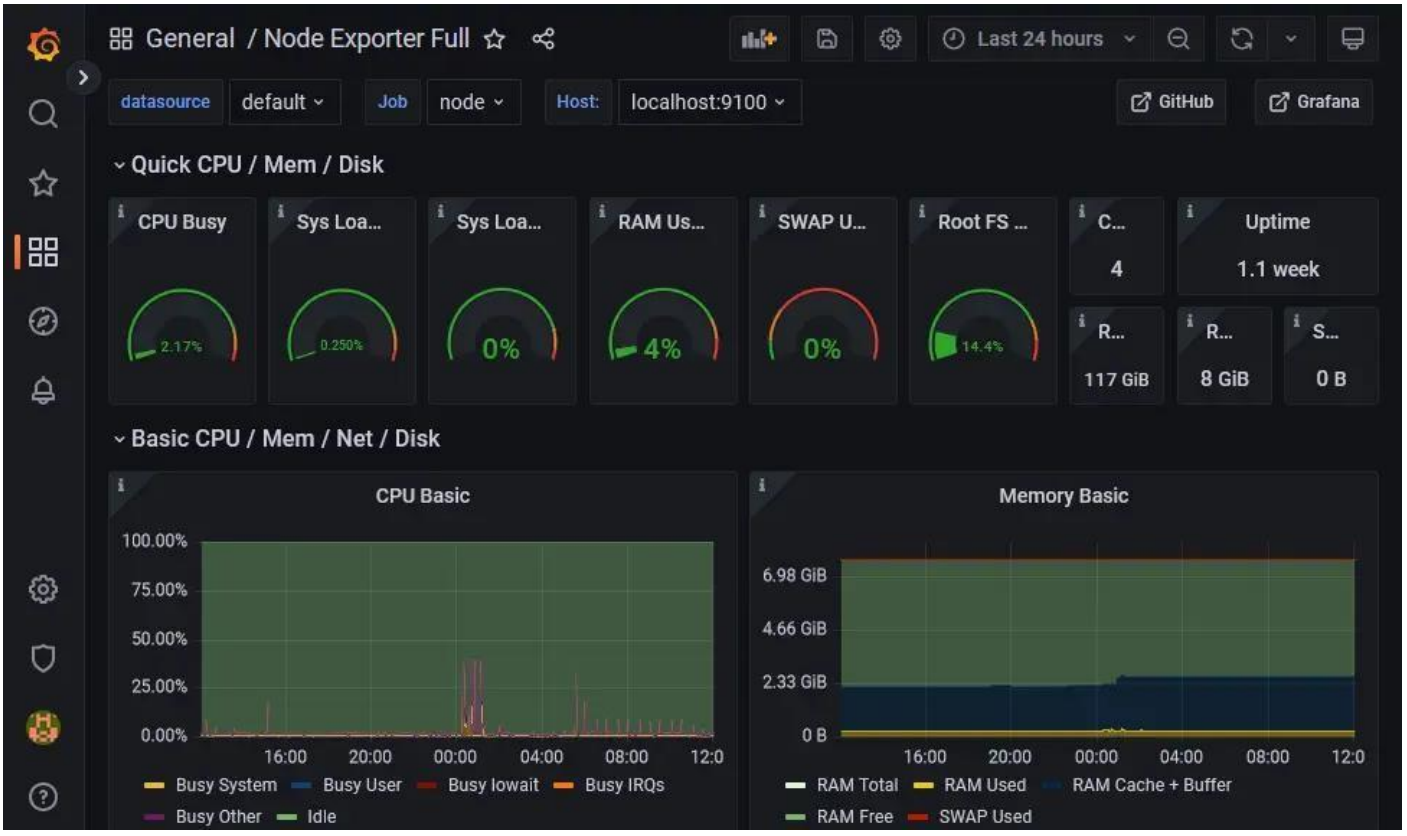


Upload the dashboard file. The finish all configuration and then create a Grafana dashboard for the master server

metrics



Finally, create a dashboard.



THE END