Production-Grade CI/CD Pipeline

GitHub → Jenkins → AWS SSM → EC2 Worker → Docker → ECR → ECS Fargate → ALB → Route53 → End User

Key Design Principles:

- **No SSH access to servers**
- **Jenkins does NOT use worker nodes as build agents**
- **Jenkins triggers deployment using AWS SSM RunCommand**
- **No SSH**
- **No passwords in Jenkins**
- **No static AWS keys**
- **IAM Roles for:**
    - **Jenkins EC2**
    - **Worker EC2**
- **SSM for all remote execution**
- **GitHub PAT stored in Secrets Manager**
- **Worker fetches secrets securely at runtime**

Architecture satisfies several PCI DSS (Payment Card Industry) and SOC2 frame.

- **PCI Requirement 1: Install and maintain a firewall configuration (Achieved via Security Groups & No SSH).**
- **PCI Requirement 7: Restrict access to data by business need-to-know (Achieved via IAM Roles and Secrets Manager).**
- **SOC2 Trust Principle (Security): The system is protected against unauthorized access (Achieved via SSM and Zero-Trust logic).**

Components Used

- GitHub – Source control
- Jenkins – CI/CD orchestrator
- AWS SSM – Secure remote command execution
- Docker – Containerization
- Amazon ECR – Container registry
- Amazon ECS (Fargate) – Container runtime
- AWS Secrets Manager – Secure GitHub PAT storage
- IAM Roles – No static credentials anywhere
- ALB
- Route53
- ACM
- EC2 Worker Node

Full Automation Flow

1) Developer pushes code to GitHub
2) GitHub Webhook triggers Jenkins
3) Jenkins pulls Jenkinsfile
4) Jenkins analyzes which service changed (monorepo logic)
5) Jenkins calls AWS SSM RunCommand
6) SSM executes `deploy.sh` on Worker EC2
7) Worker:

a) Pulls latest code

b) Builds new Docker image (no cache)

c) Tags with timestamp version

d) Pushes image to ECR

e) Fetches current ECS task definition

f) Updates image using `jq`

g) Registers new task revision

h) Updates ECS service

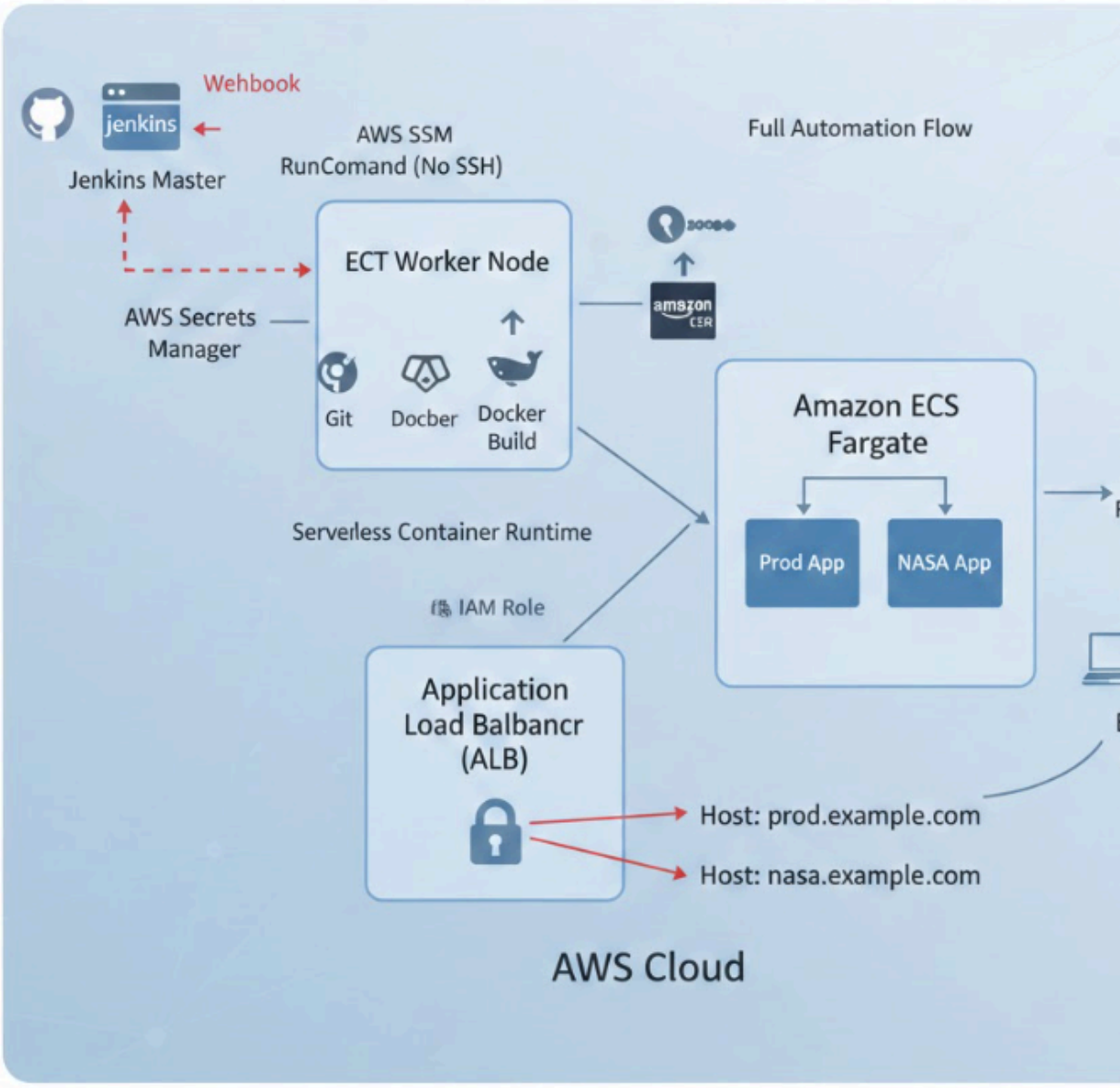
i) Waits for service to stabilize

8) ECS performs rolling update

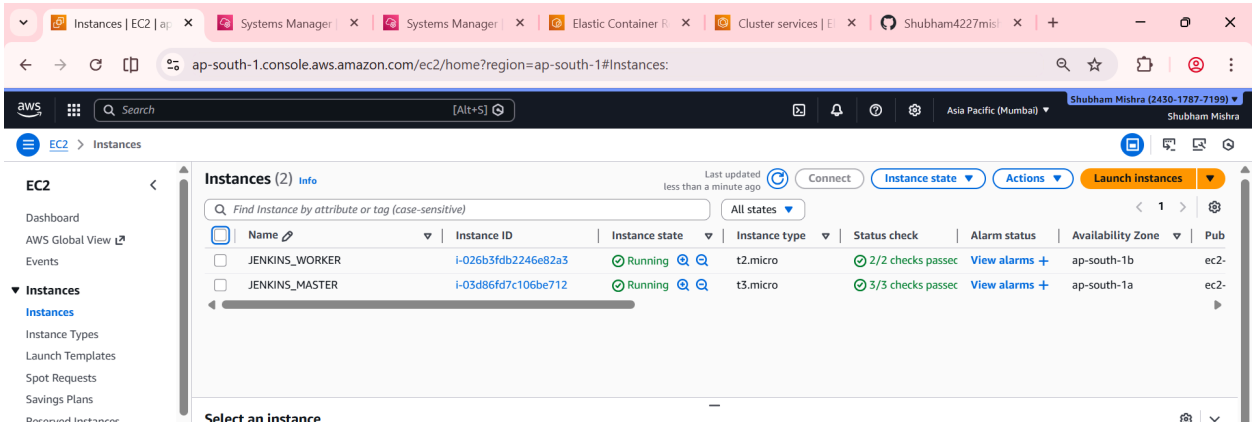
9) ALB continues serving traffic with zero downtime

10) Users see new version live.

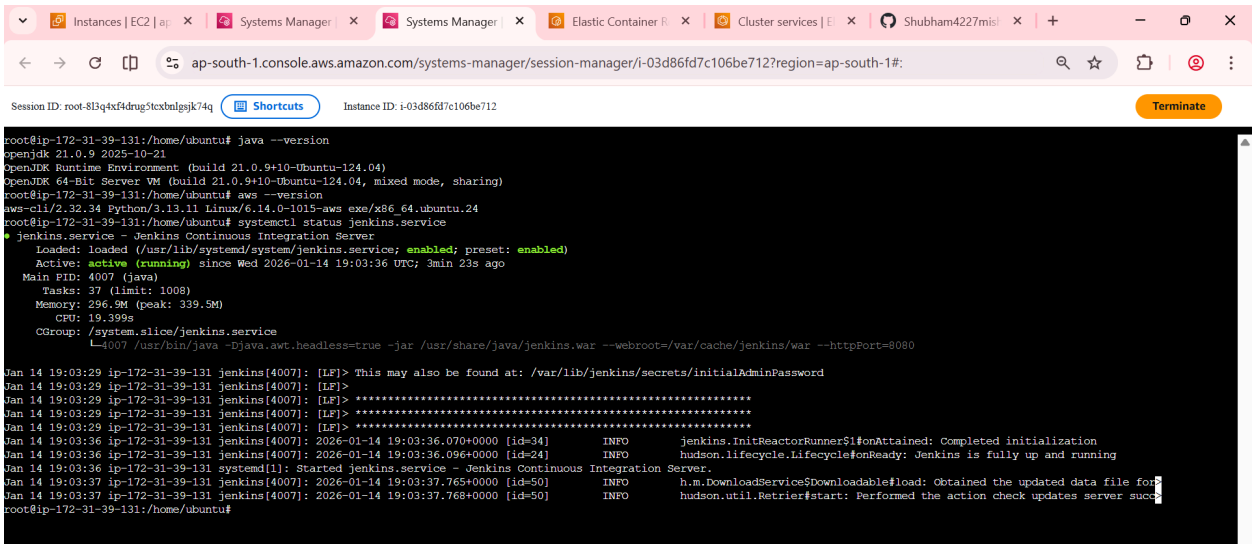
Zero-Trust, SSM-Driven CI/CD Pipeline



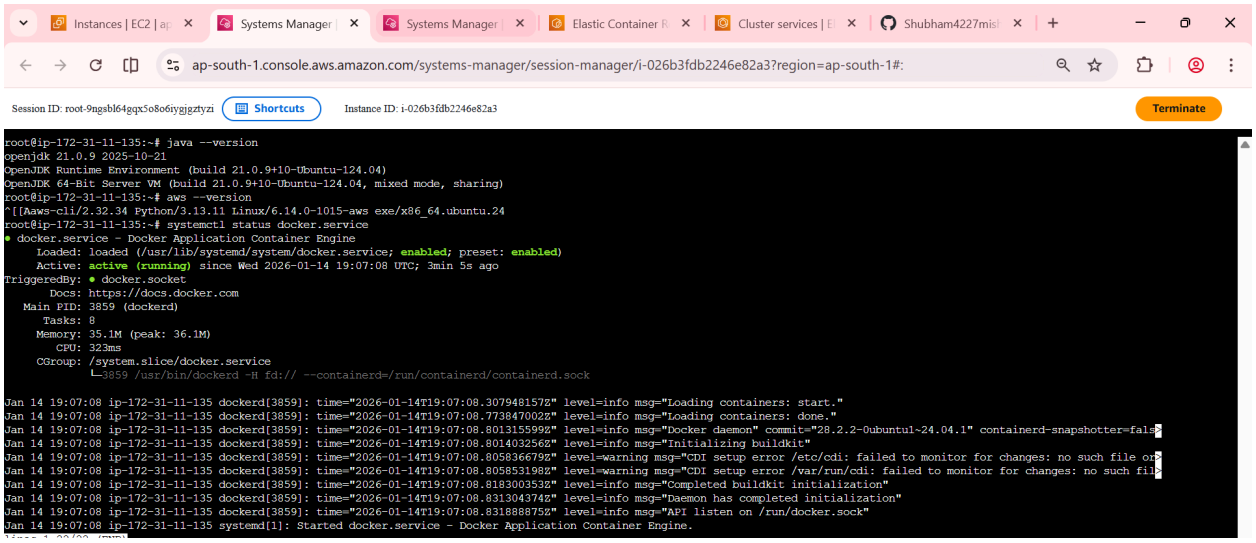
Step 1) We will launch two ec2 instances 1 will act as jenkins master and other will act as worker node



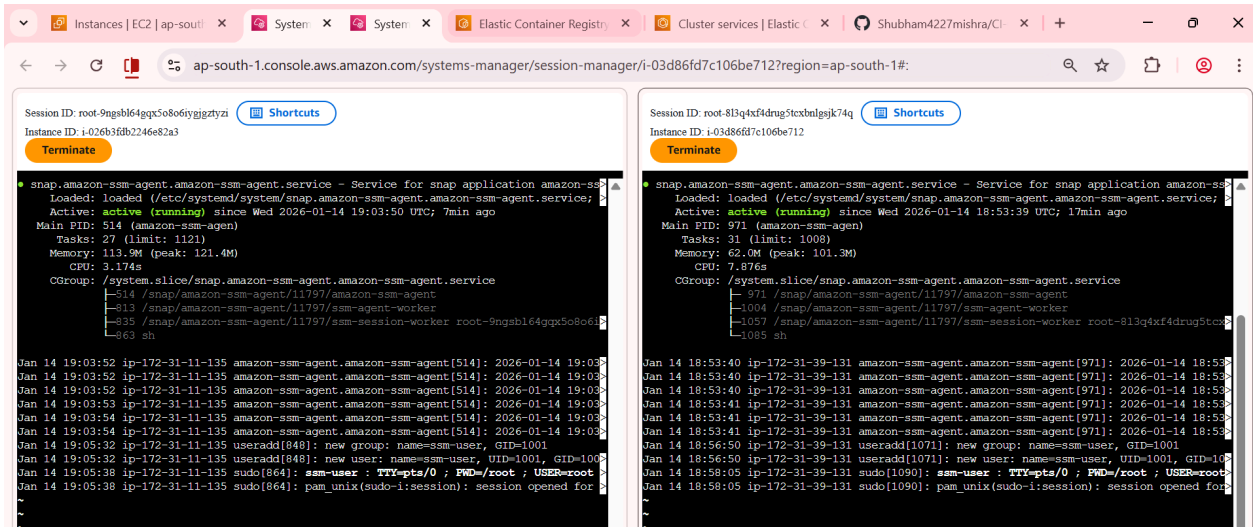
Step 2) On jenkins master we installed java, jenkins and awscli



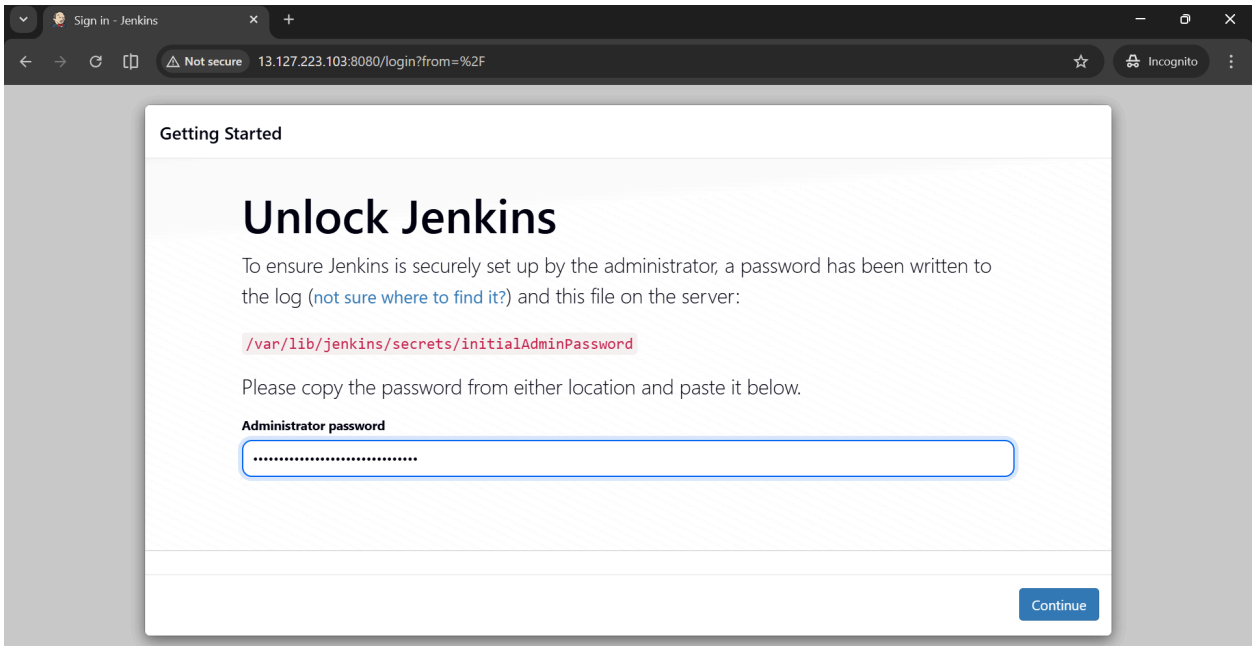
Step 3) Now on worker ec2 we will install java, awscli and docker



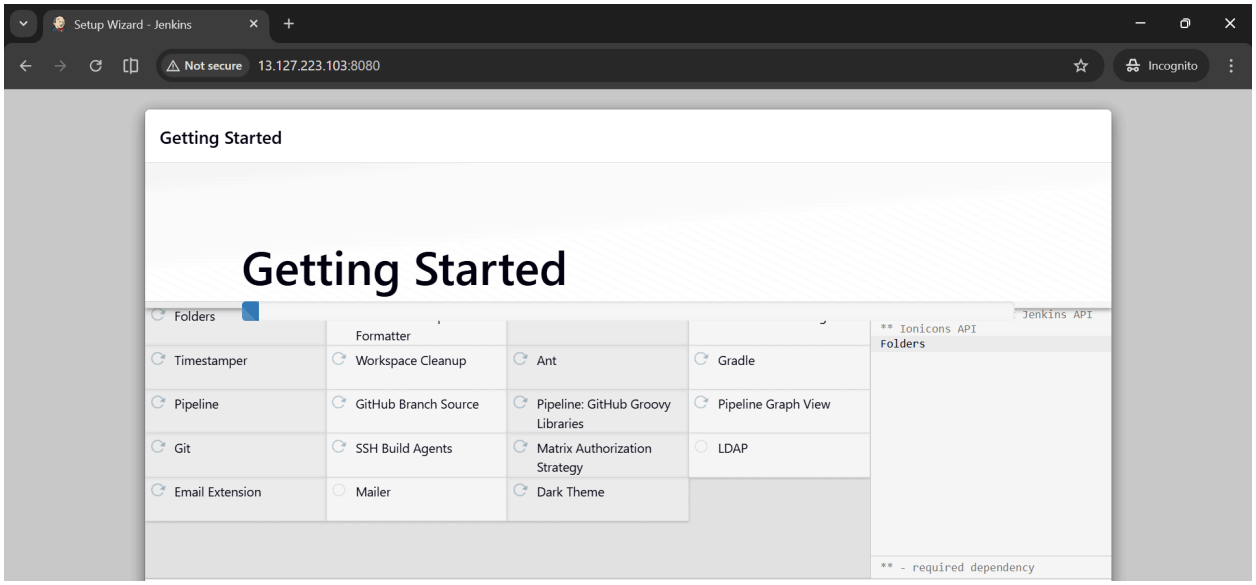
Step 4) Now on both master and worker node ssm agent is running as we wont use ssh or pem key for robust security



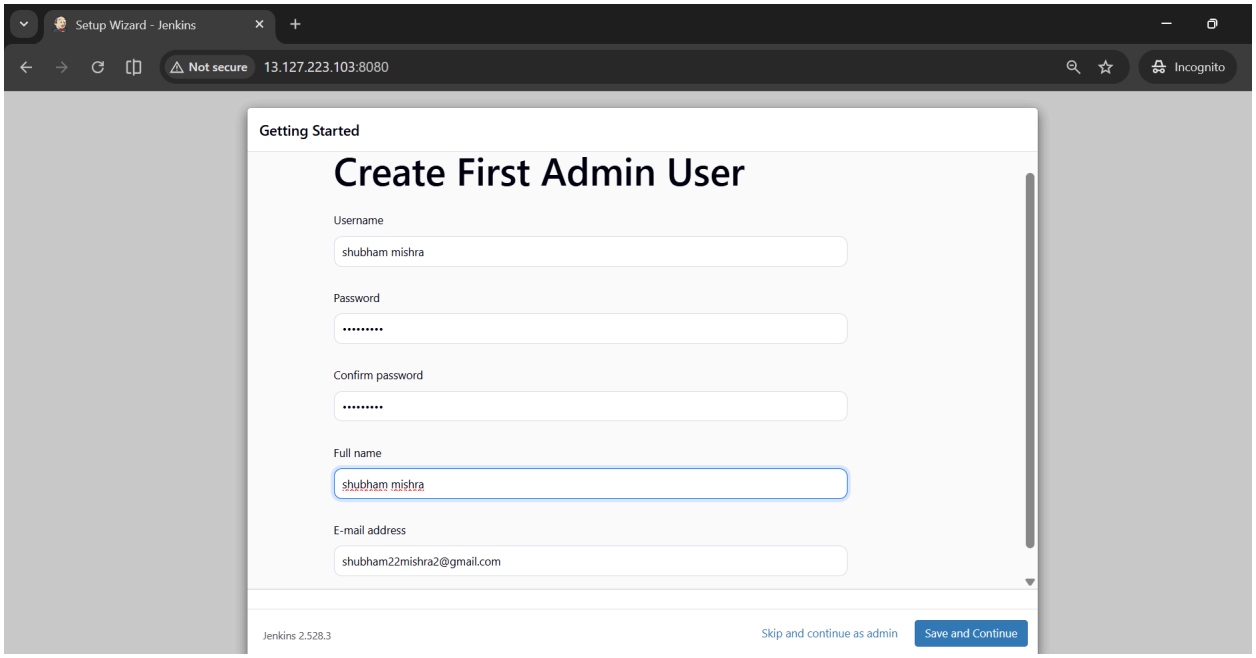
Step 5) Now we will setup jenkins on master node



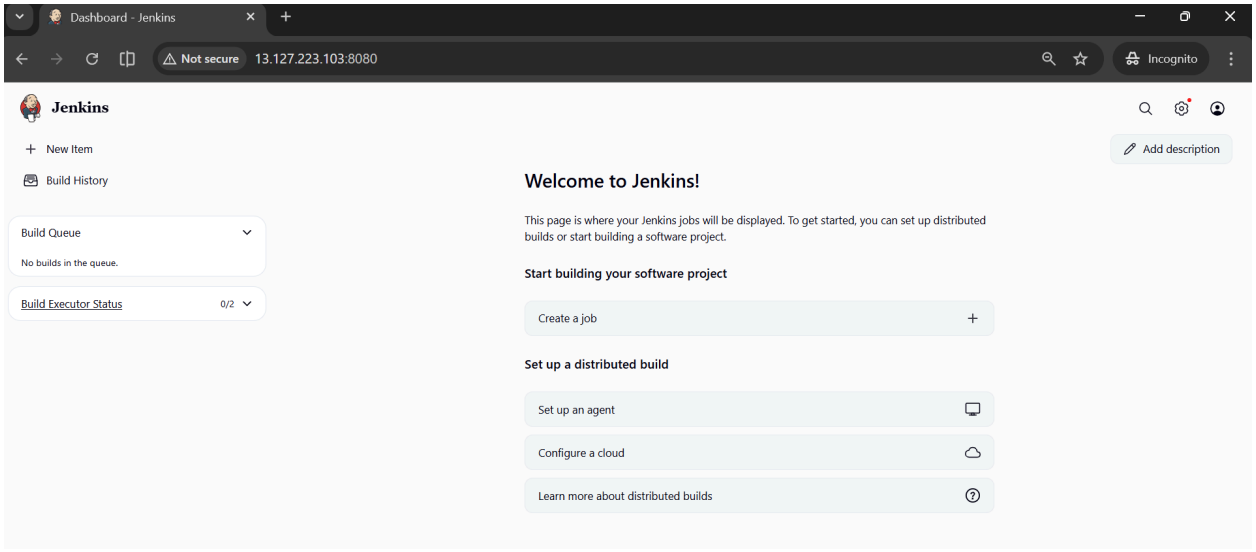
Step 6) Installing all suggested plugins on jenkins



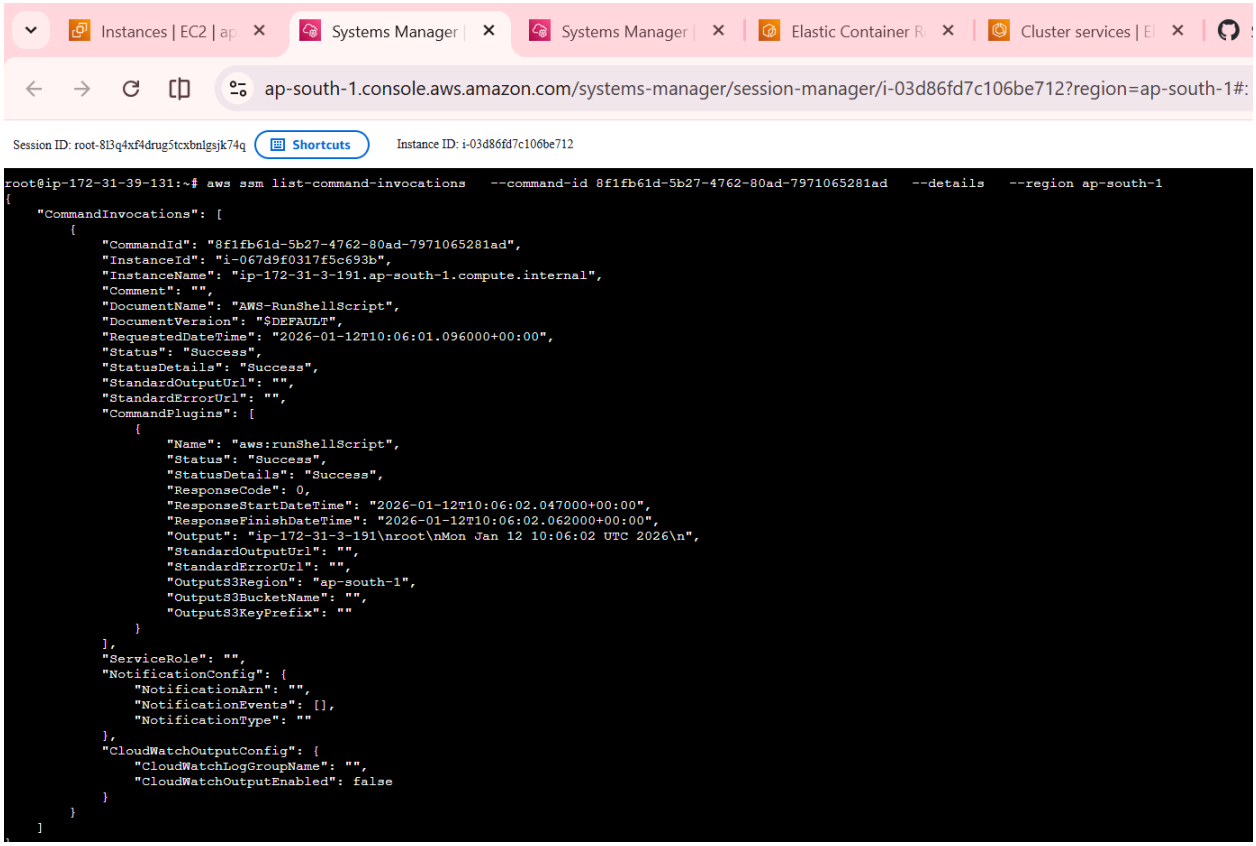
Step 7) We will setup username and password in jenkins



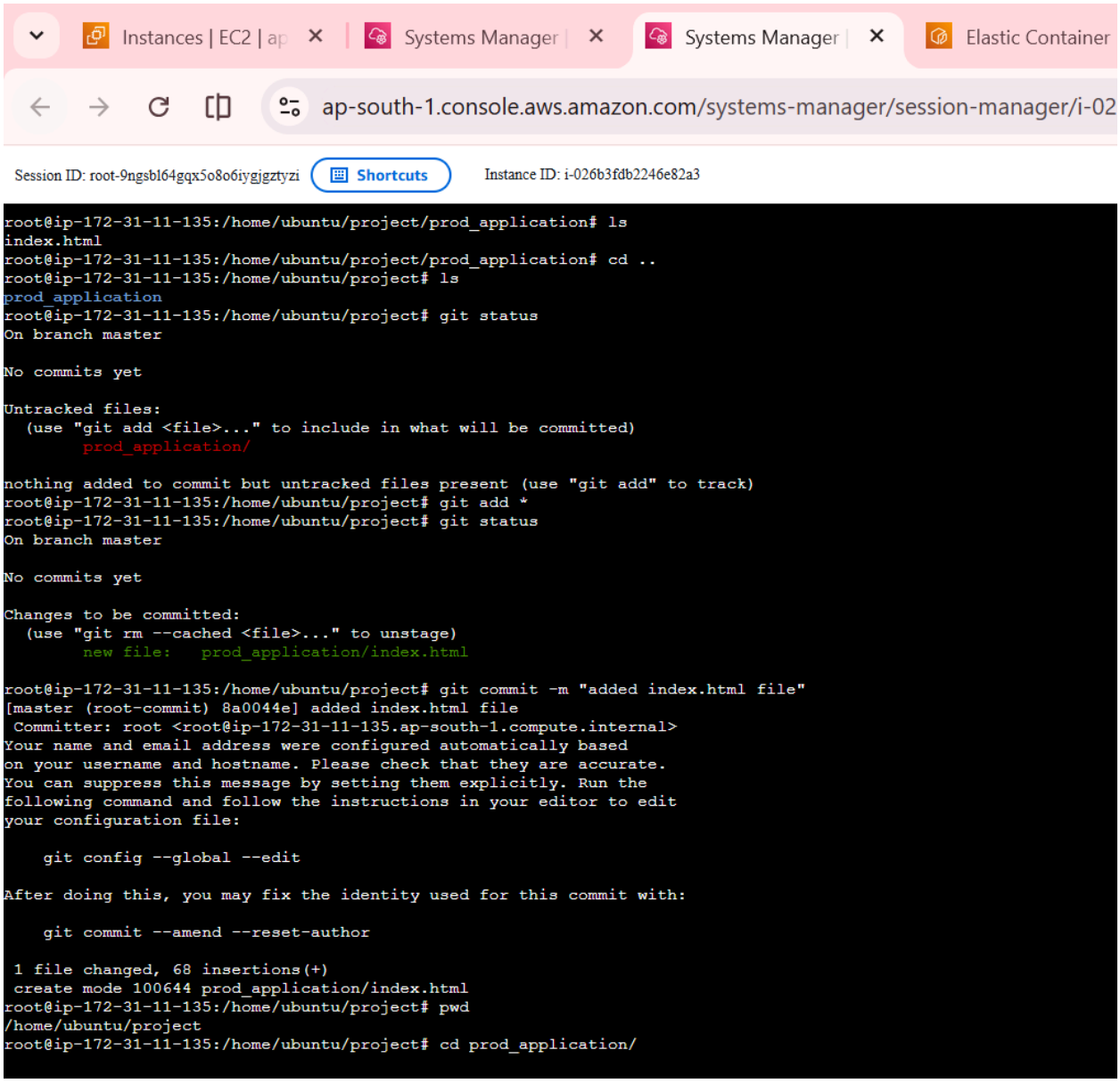
Step 8) jenkins setup successfull



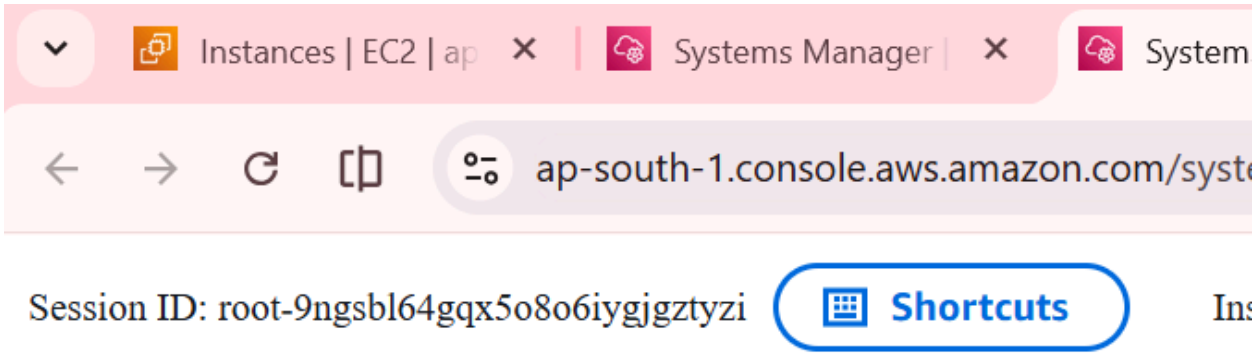
Step 9) Now we will check whether jenkins master can communicate with worker using ssm commands



Step 10) Now we will create project folder and initialise git repo also create index.html file in folder /home/ubuntu/project/prod_application

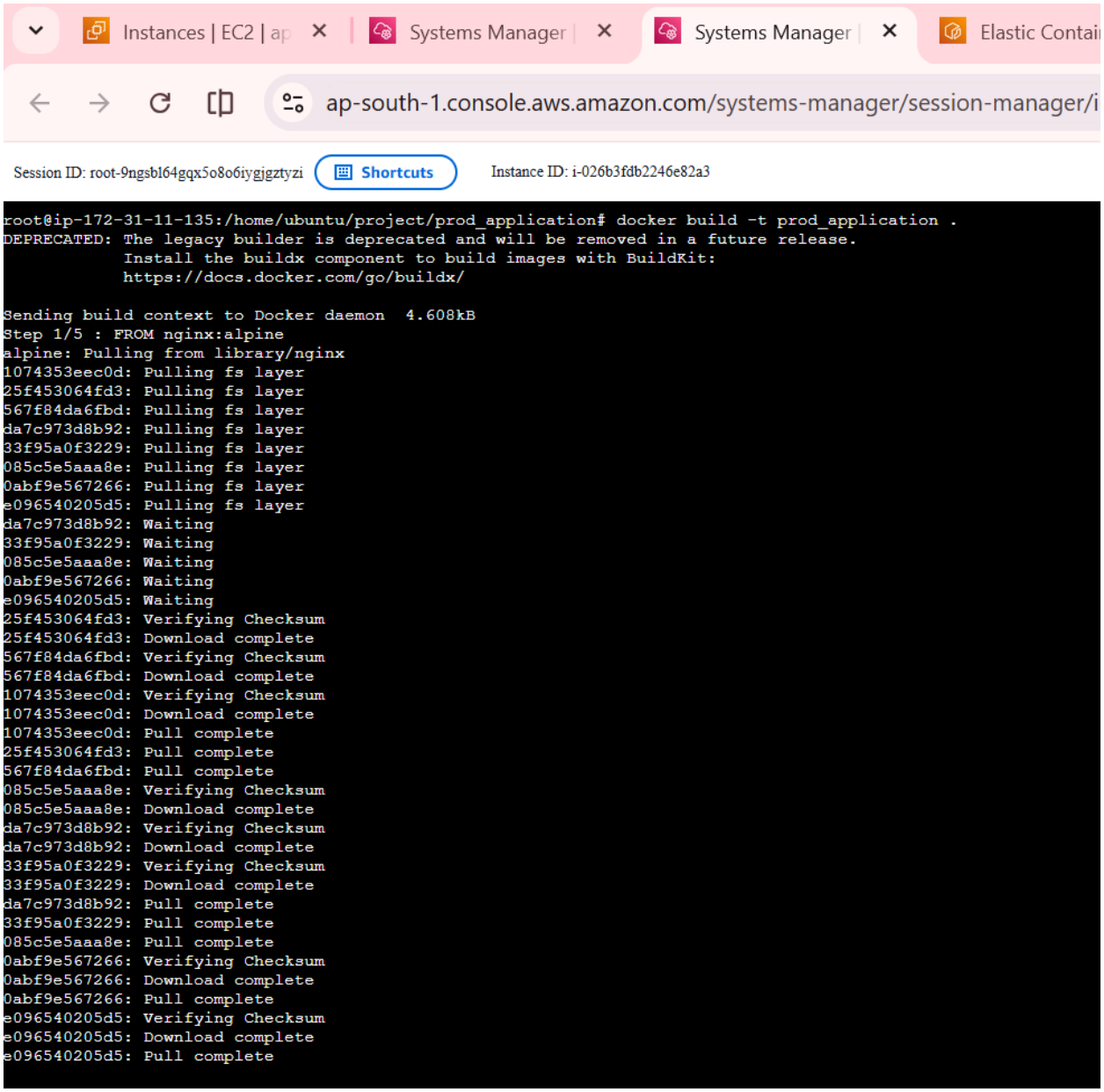


Step 11) Now we will create Docker file for this index.html code on worker node

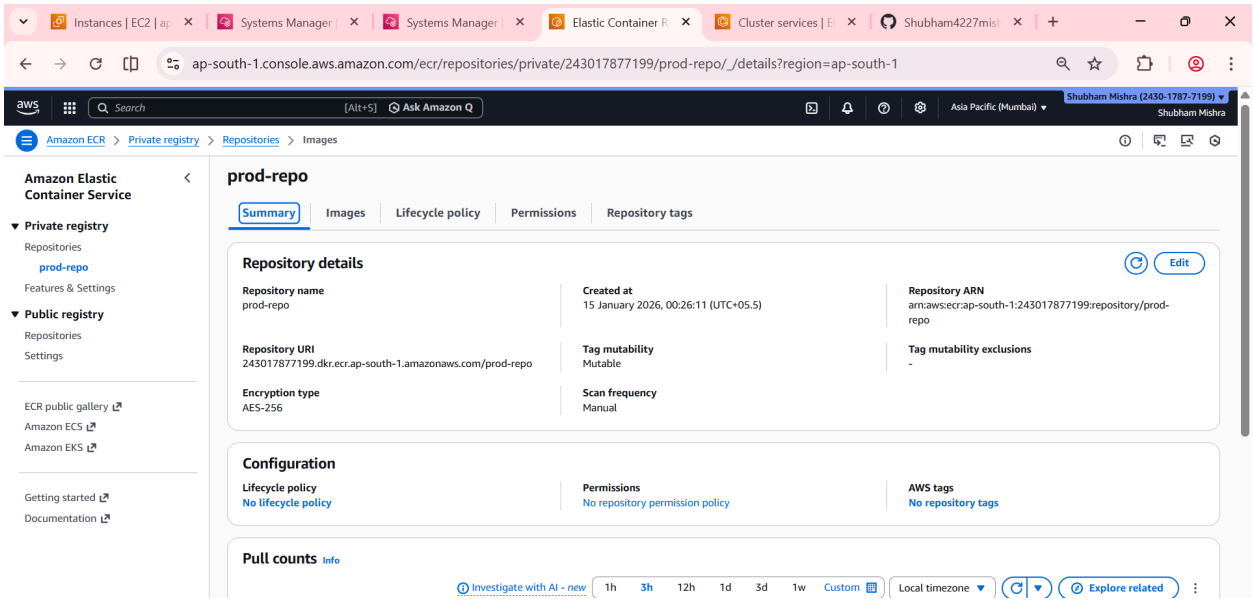



```
FROM nginx:alpine

# Remove default nginx website
RUN rm -rf /usr/share/nginx/html/*

# Copy our production page
COPY index.html /usr/share/nginx/html/index.html

# Expose port 80
EXPOSE 80

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

~
~
~
```

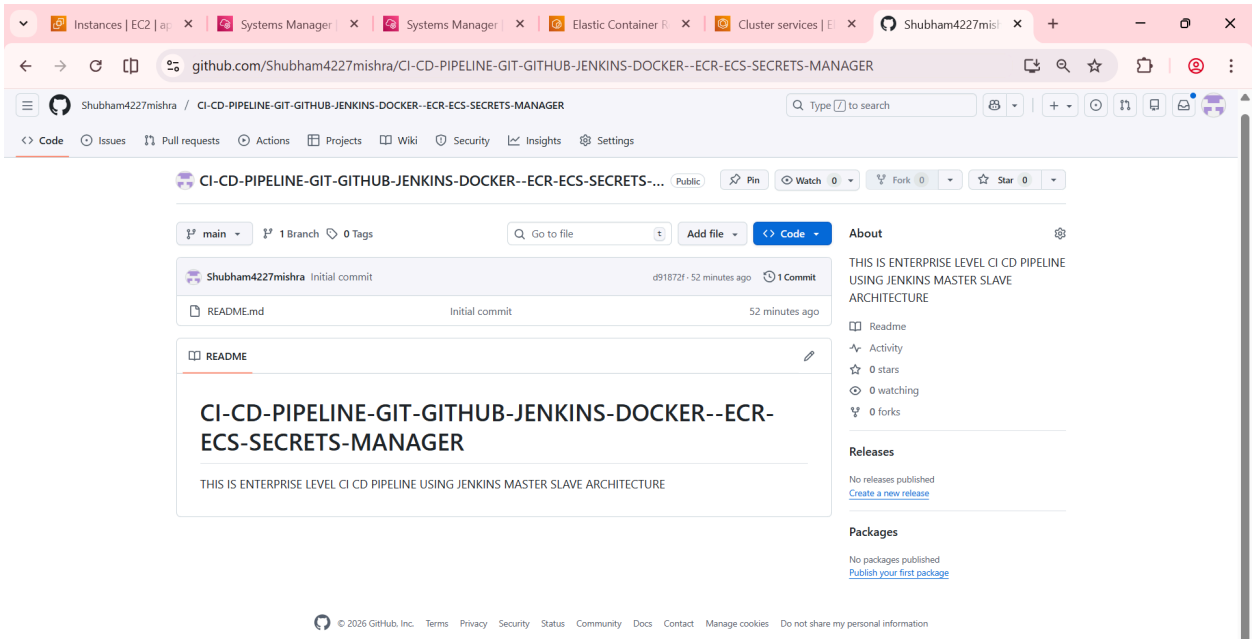
Step 12) Now we will build docker file locally on worker node



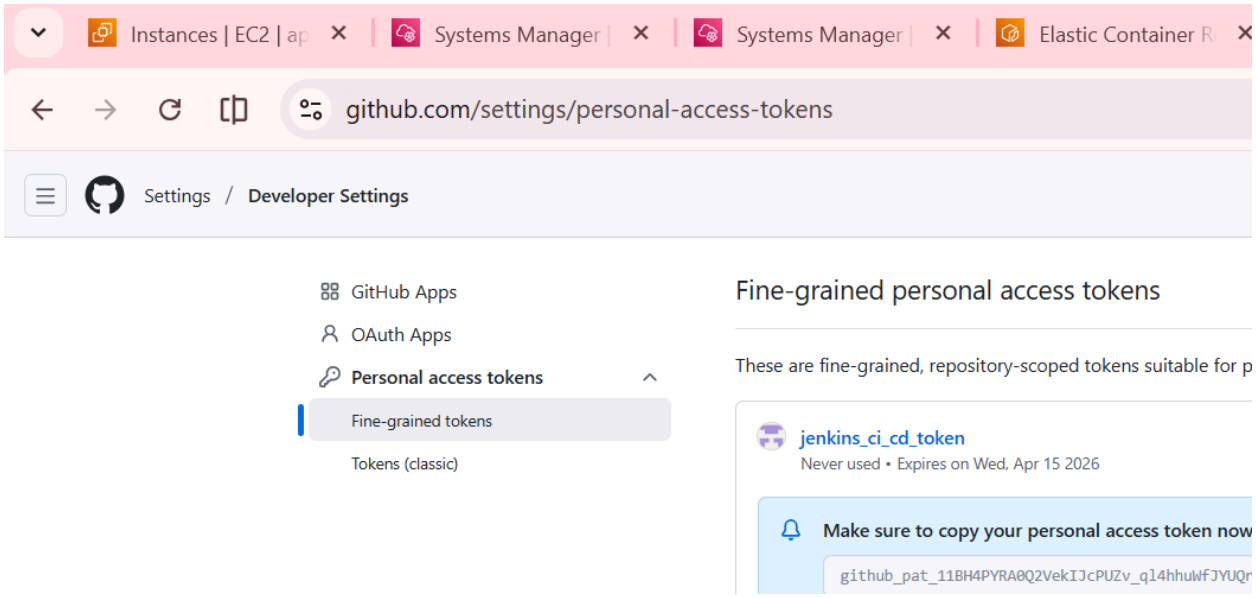
Step 13) Now we will create ECR private repo



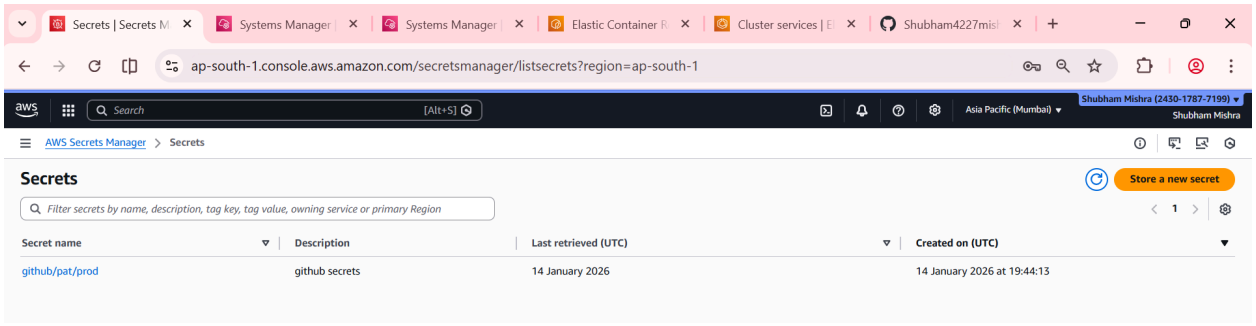
Step 14) Now we will also create github repository



Step 15) Now we will create PAT token to allow git on worker ec2 to connect to GITHUB and push code



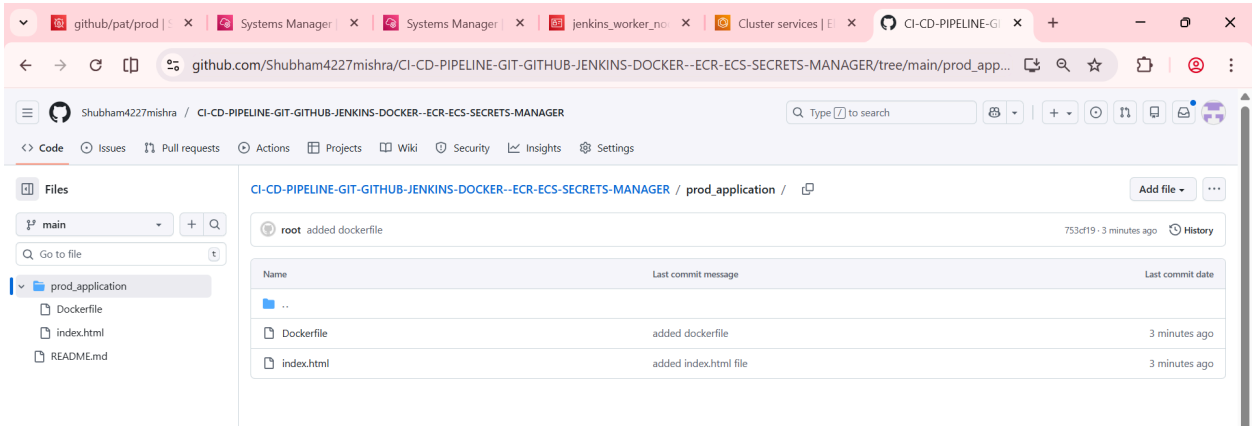
Step 16) Now we will store this pat token and username in aws secrets so that git on ec2 can use that and push code into github repo



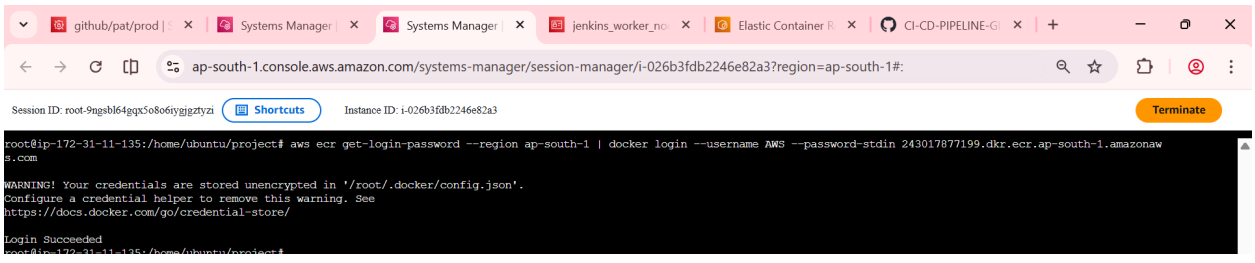
Step 17) Now we pushed our index.html and Dockerfile on github successfully

```
From https://github.com/Shubham4227mishra/CI-CD-PIPELINE-GIT-G
 * branch            main       -> FETCH_HEAD
Successfully rebased and updated refs/heads/main.
root@ip-172-31-11-135:/home/ubuntu/project# git push -u origin
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (8/8), 1.73 KiB | 1.73 MiB/s, done.
Total 8 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Shubham4227mishra/CI-CD-PIPELINE-GIT-GIT
   d91872f..753cf19  main -> main
branch 'main' set up to track 'origin/main'.
root@ip-172-31-11-135:/home/ubuntu/project#
```

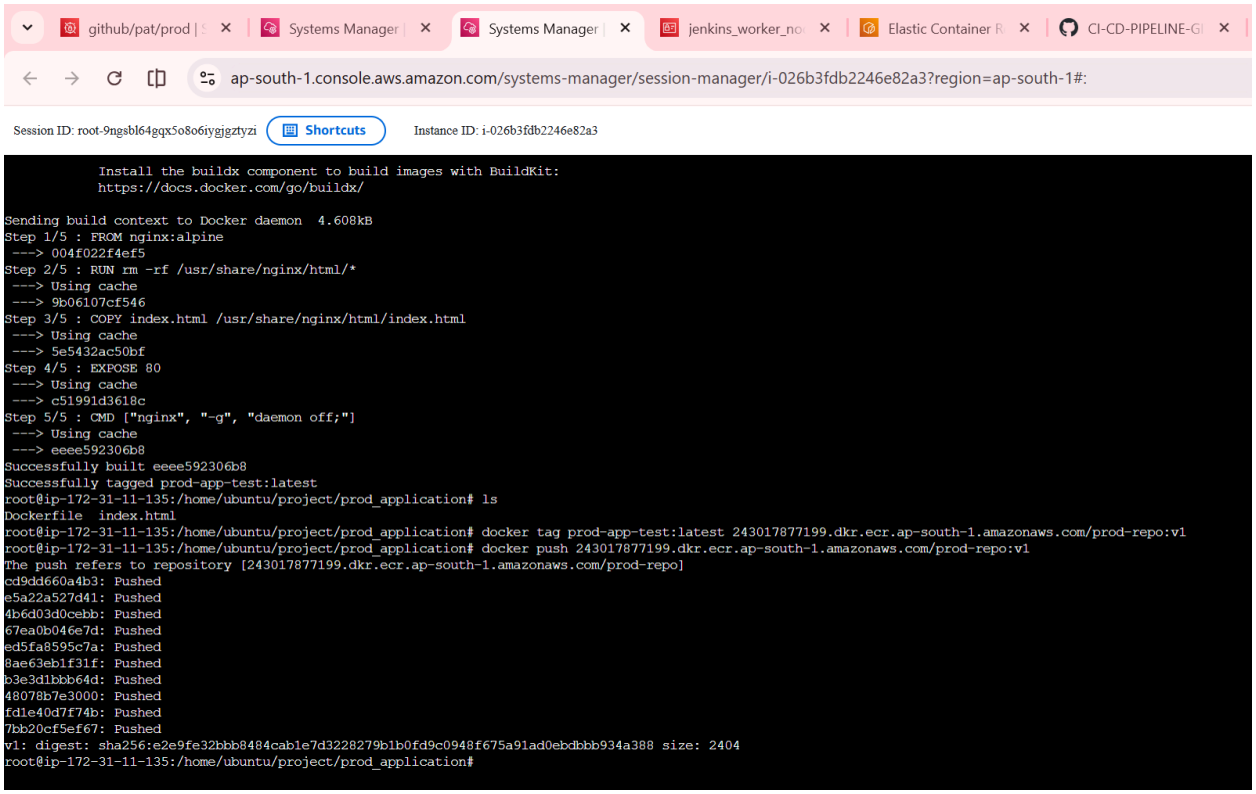
Step 18) We can verify files in github



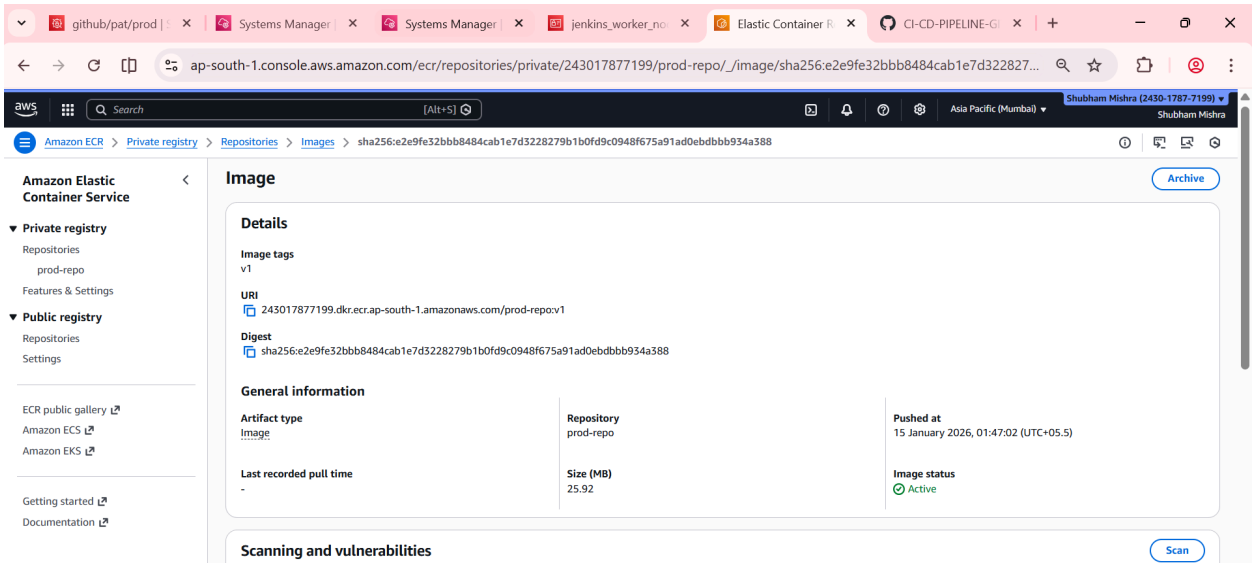
Step 19) Now from worker node we will login into ECR private repo



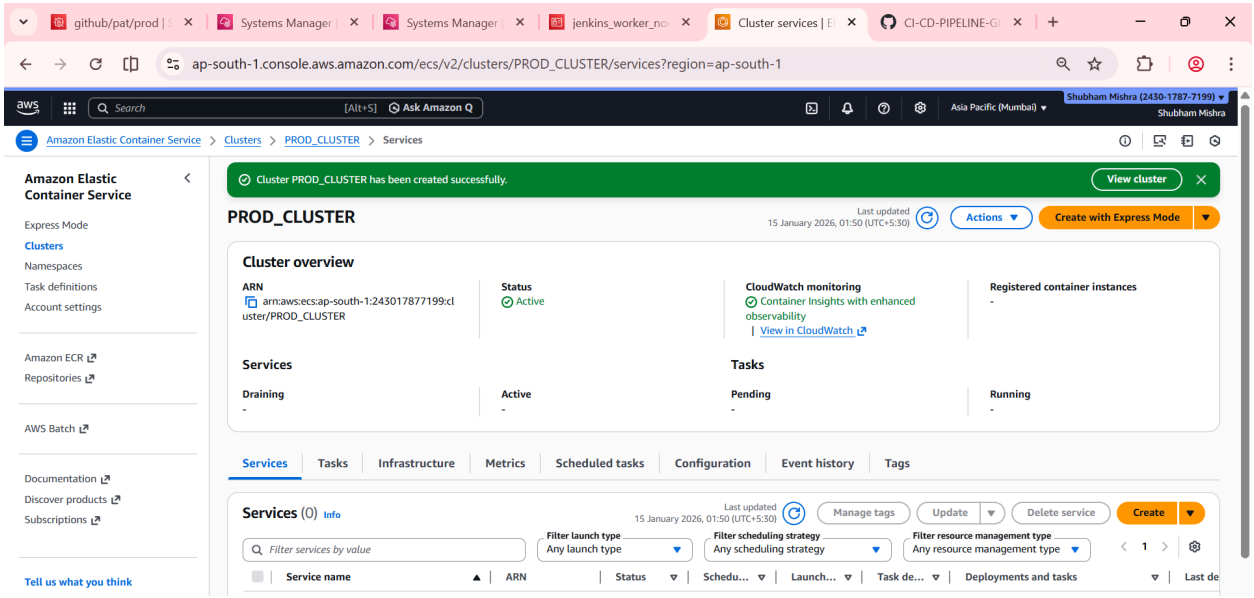
Step 20) We will build docker image and push it into ecr



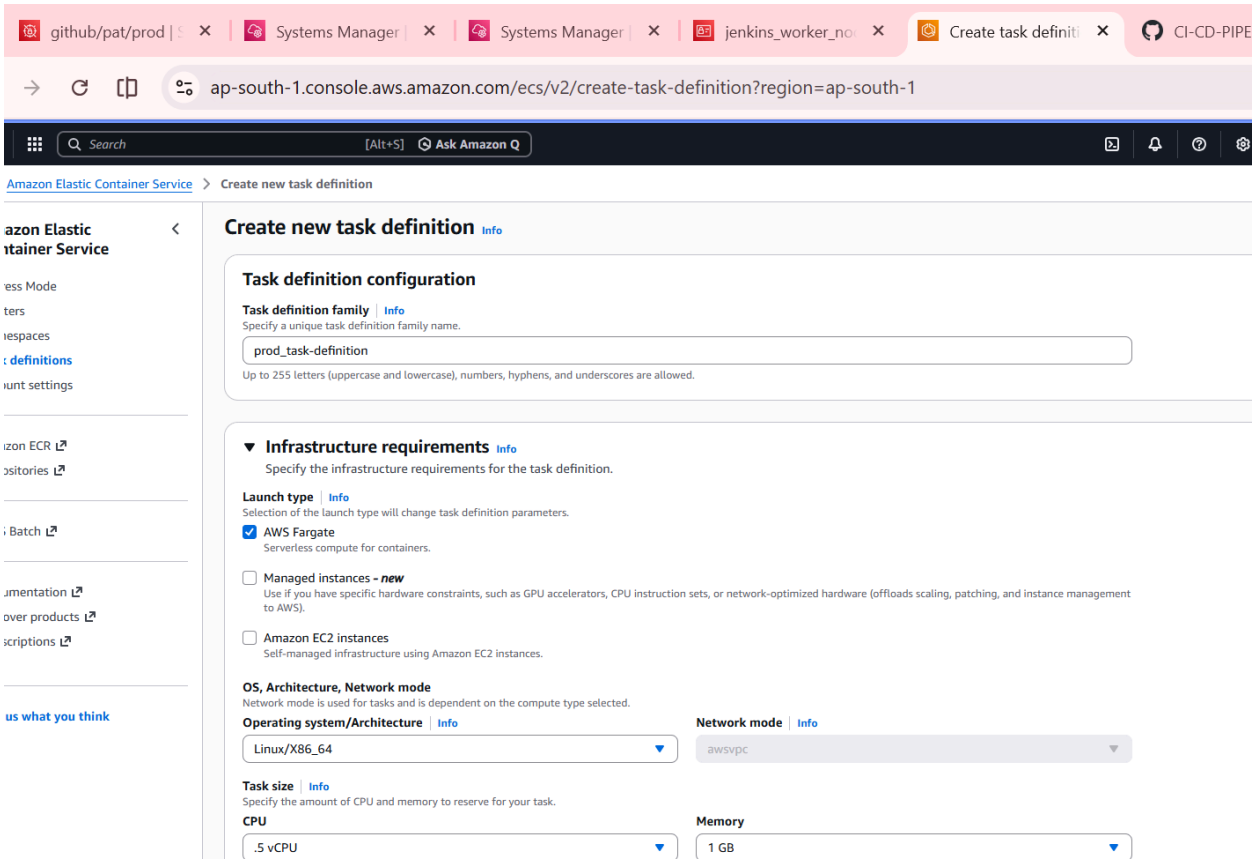
Step 21) We can verify image is pushed into ECR

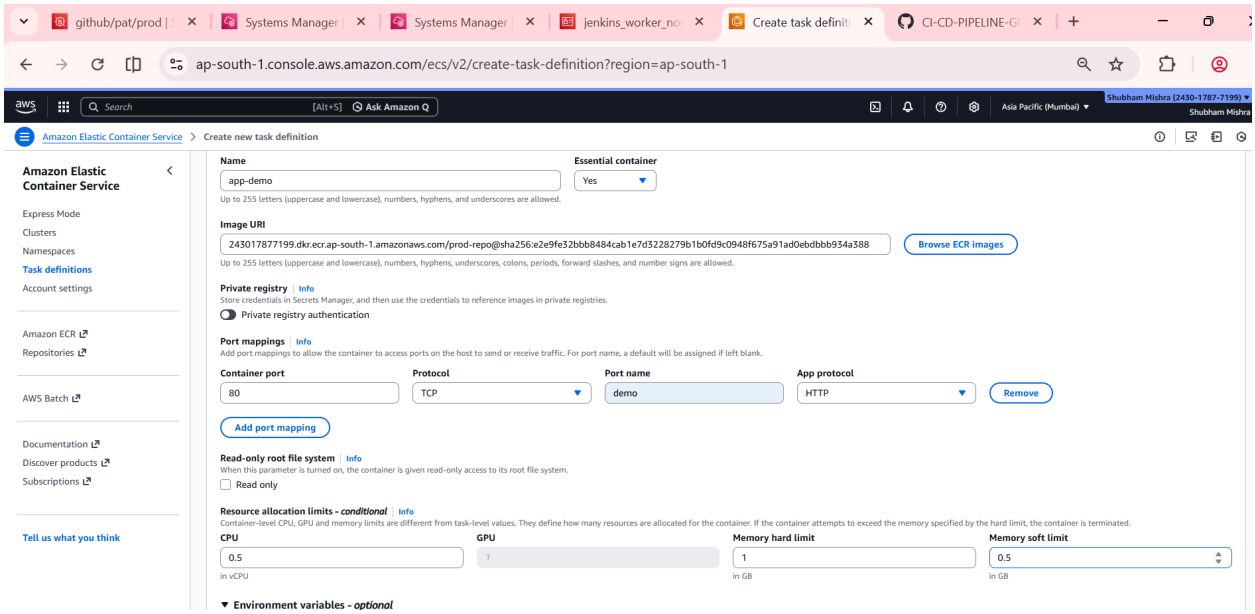


Step 22) Now we will create ECS cluster

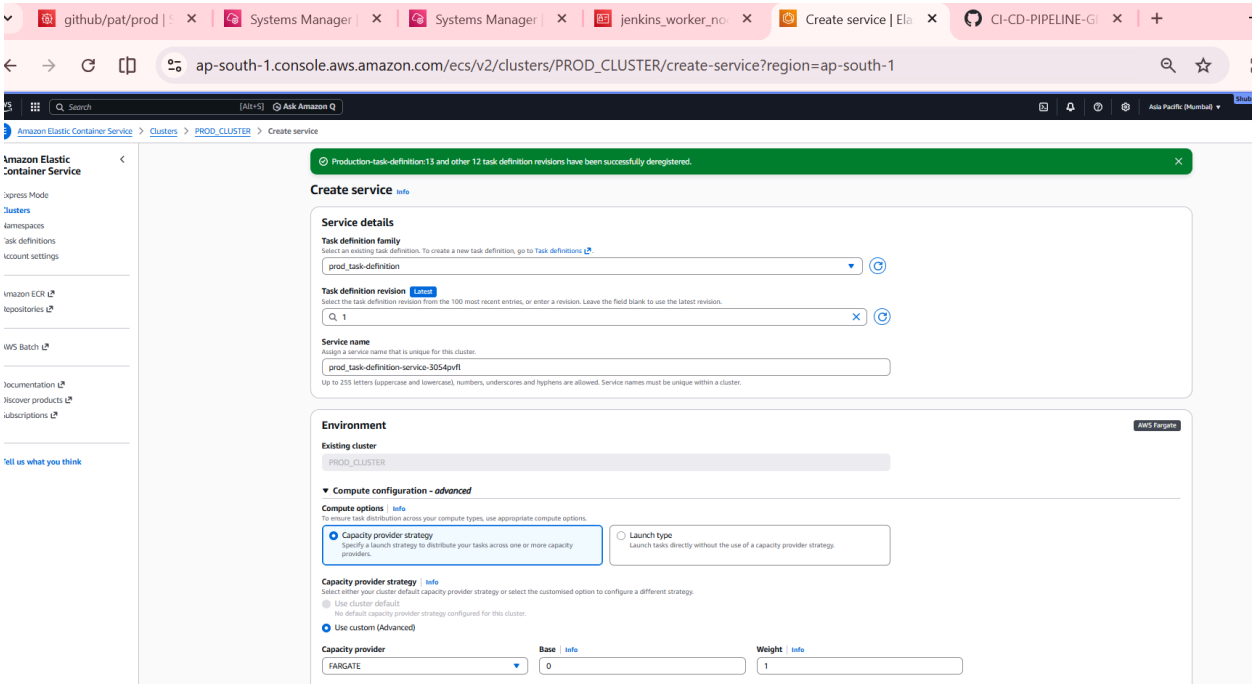


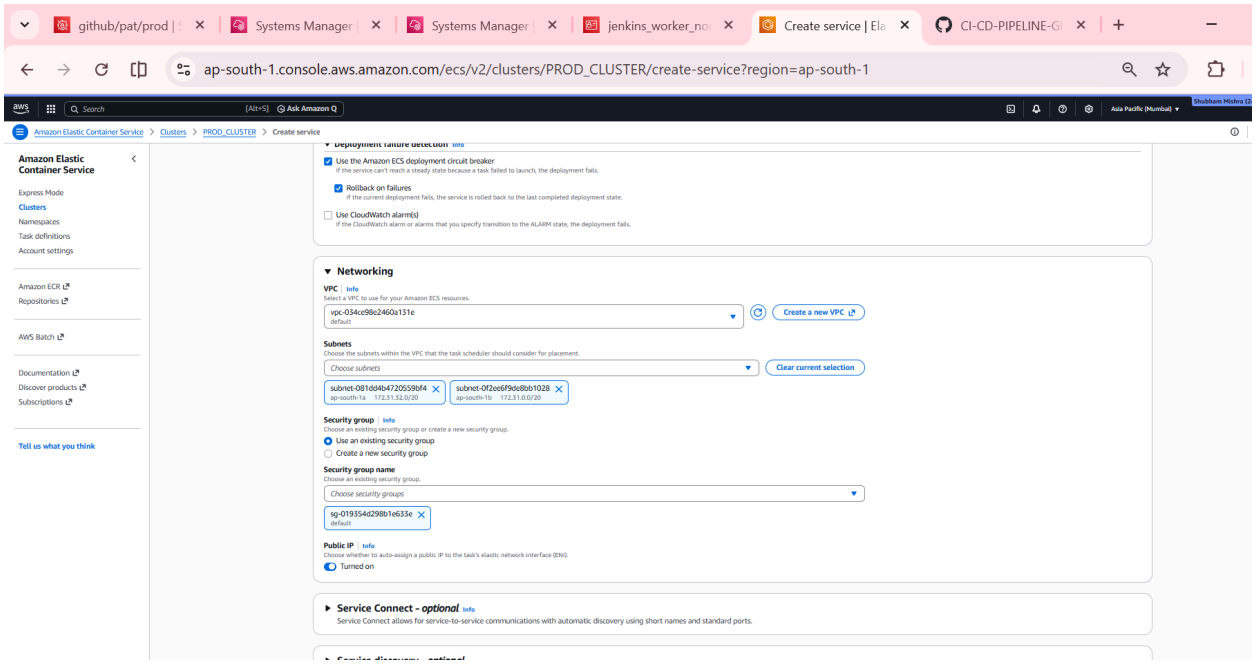
Step 23) Now we will create task definition for the cluster



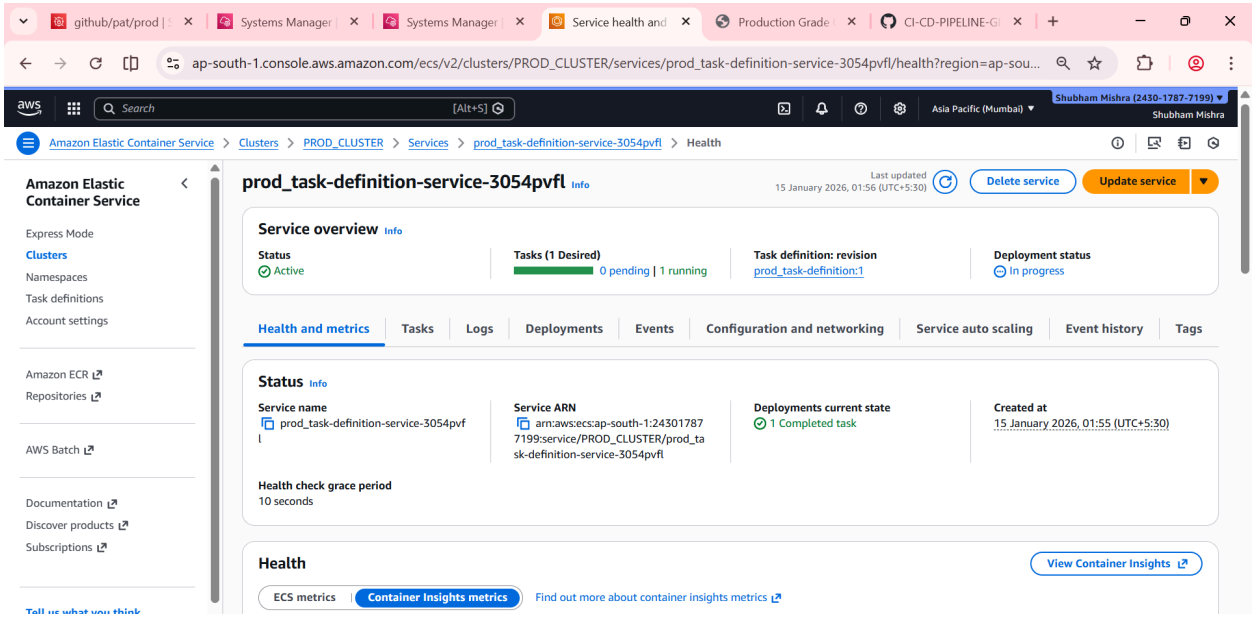


Step 24) Now we will create service for the task definition and cluster

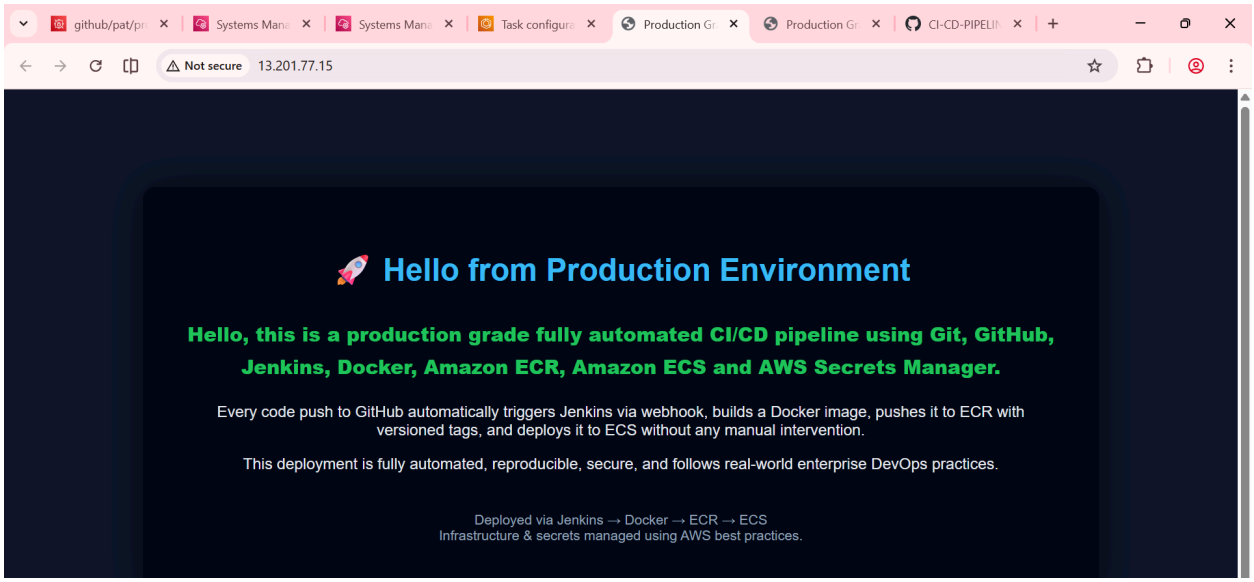




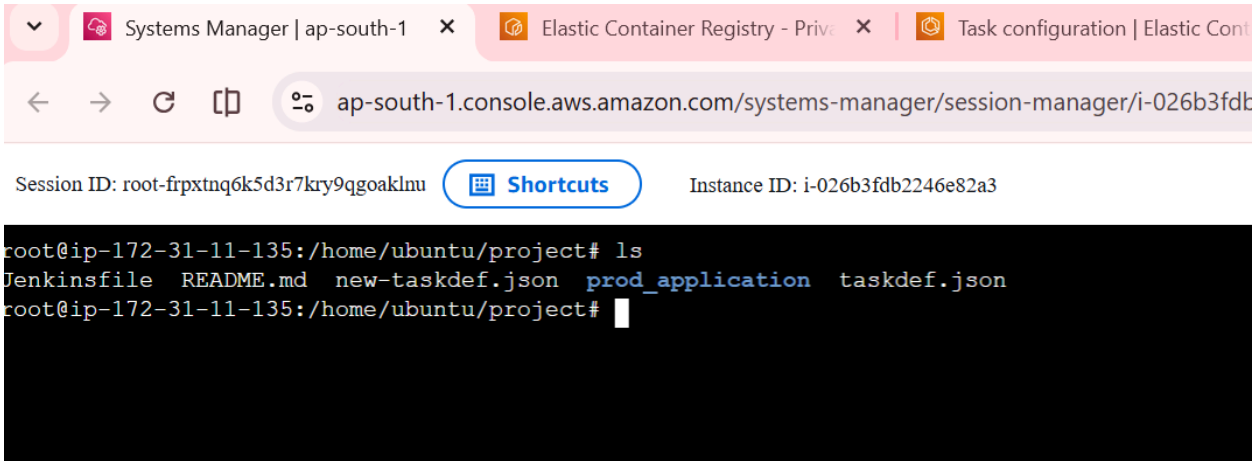
Step 25) Now we can see the deployment is in progress



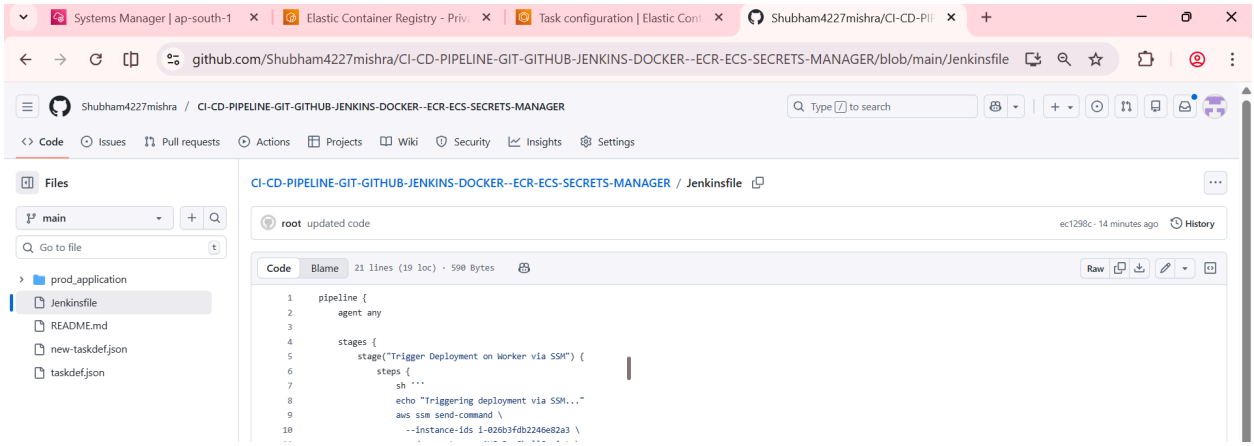
Step 26) Now after successful deployment we can see 1 container having private ip is launched and we can verify it



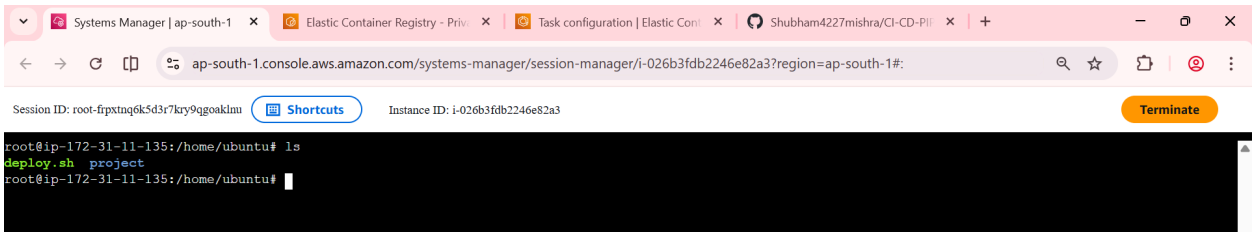
Step 27) Now we will automate this deployment on every code changes so we will create jenkins file on worker node and push it on github



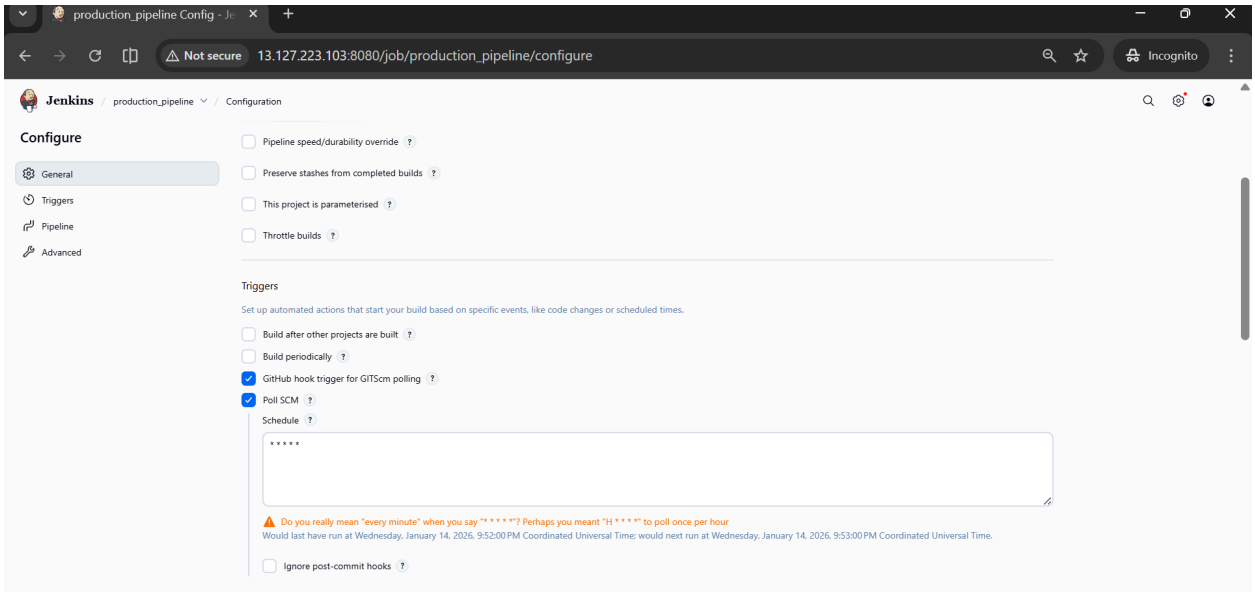
Step 28) Now Jenkinsfile is successfully pushed on github as jenkins will use this file to build the ci cd pipeline automatically

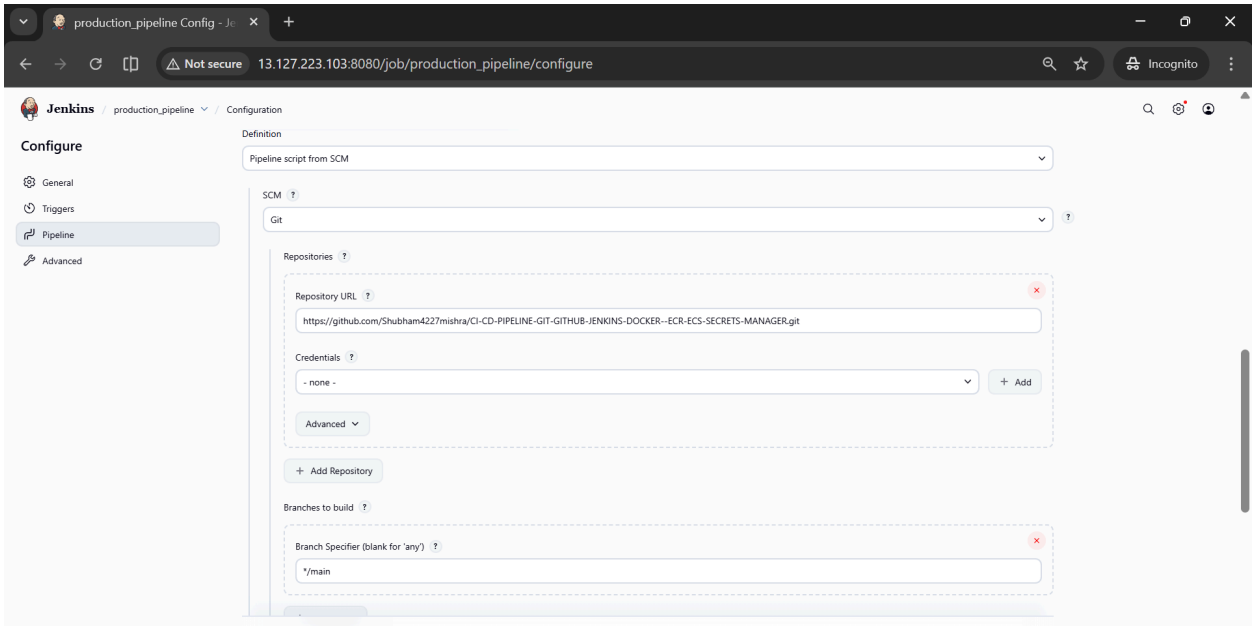


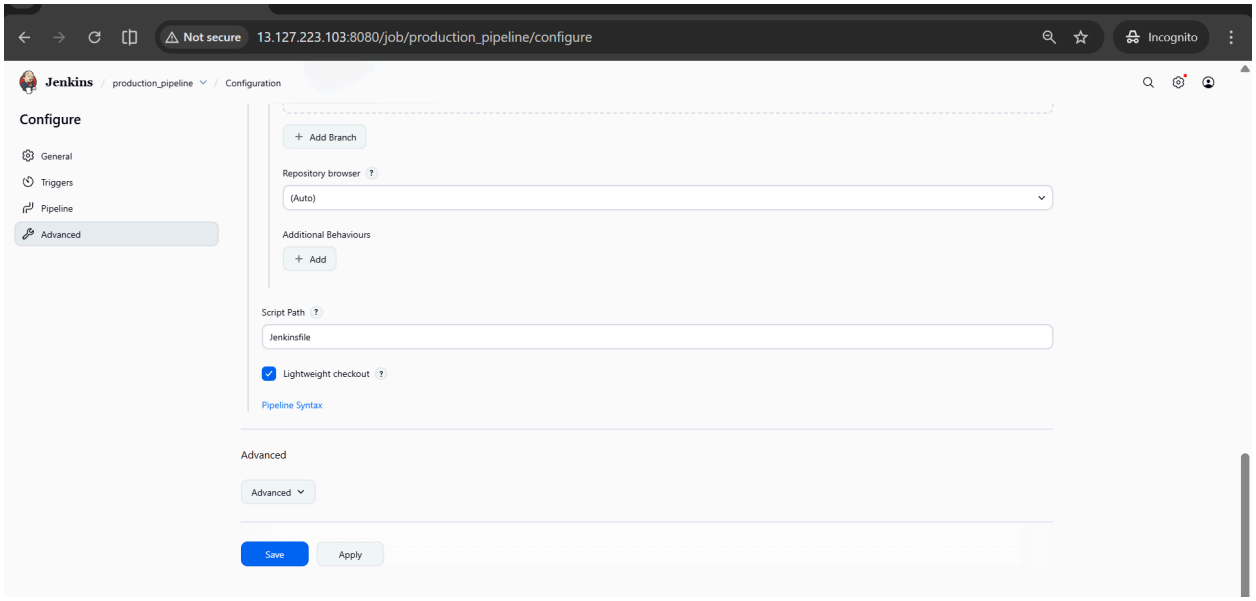
Step 29) Now we will create deploy.sh file on worker ec2 as we want master to use this file to deploy jobs on worker node only not on itself



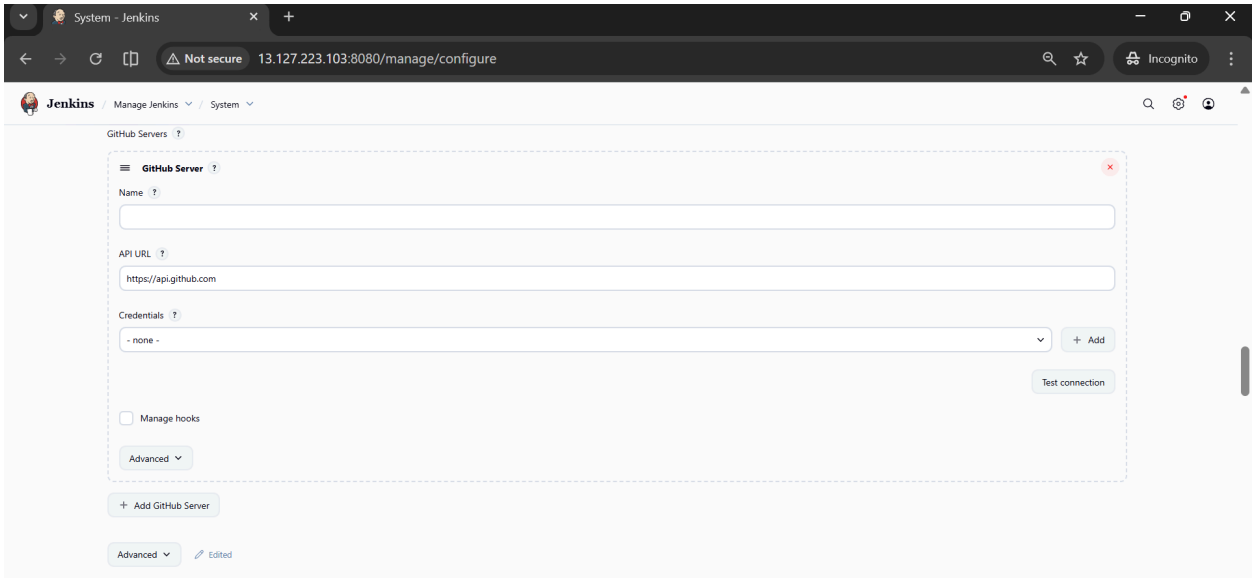
Step 30) Now we wil create job in jenkins



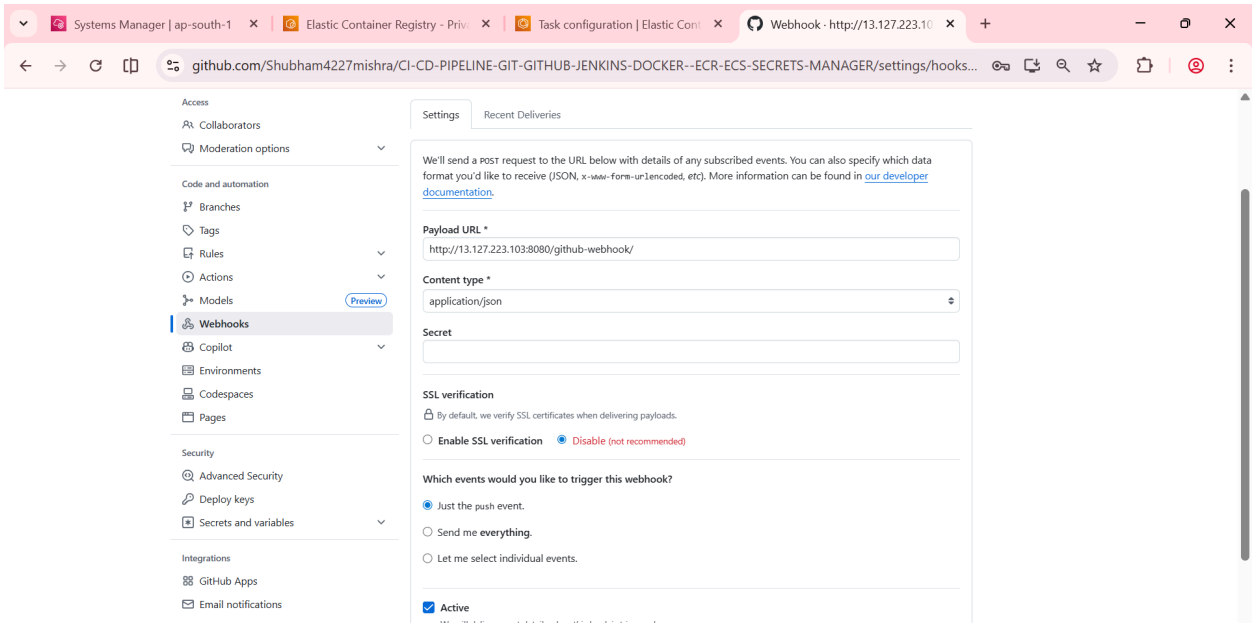




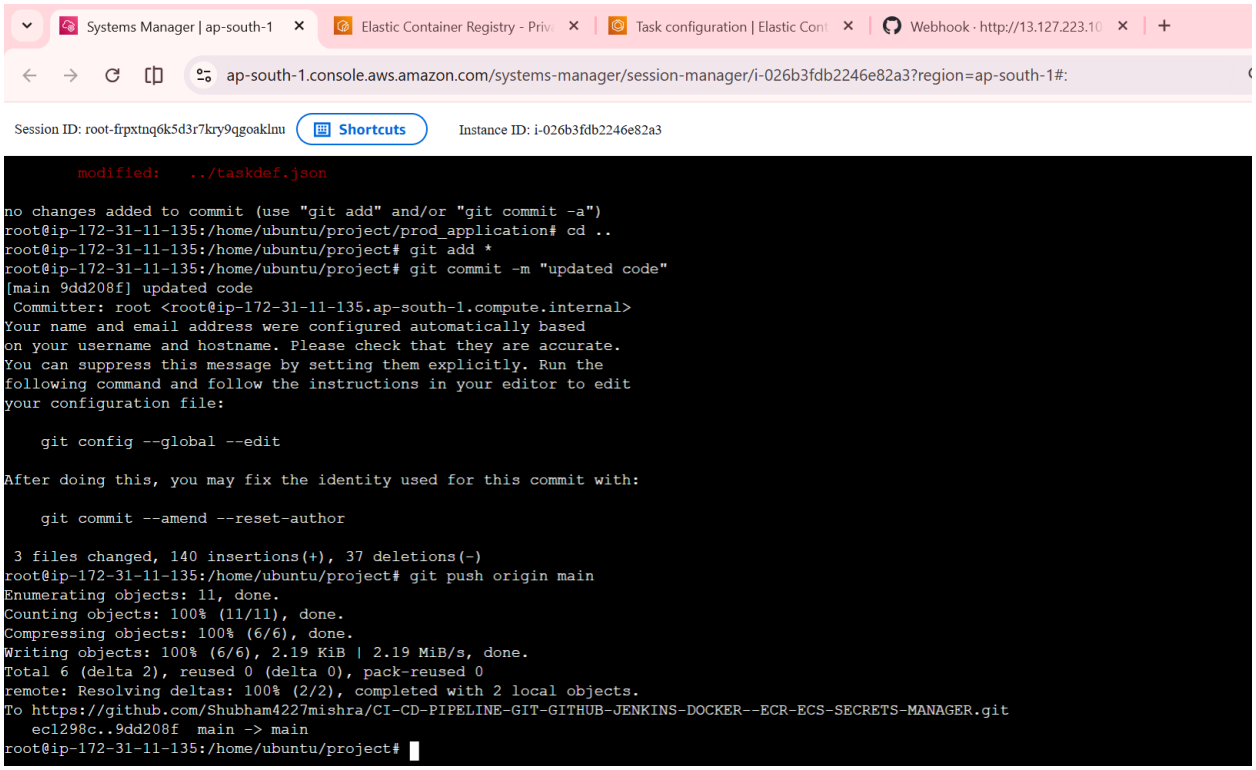
Step 31) Also make sure this is configured in jenkins system



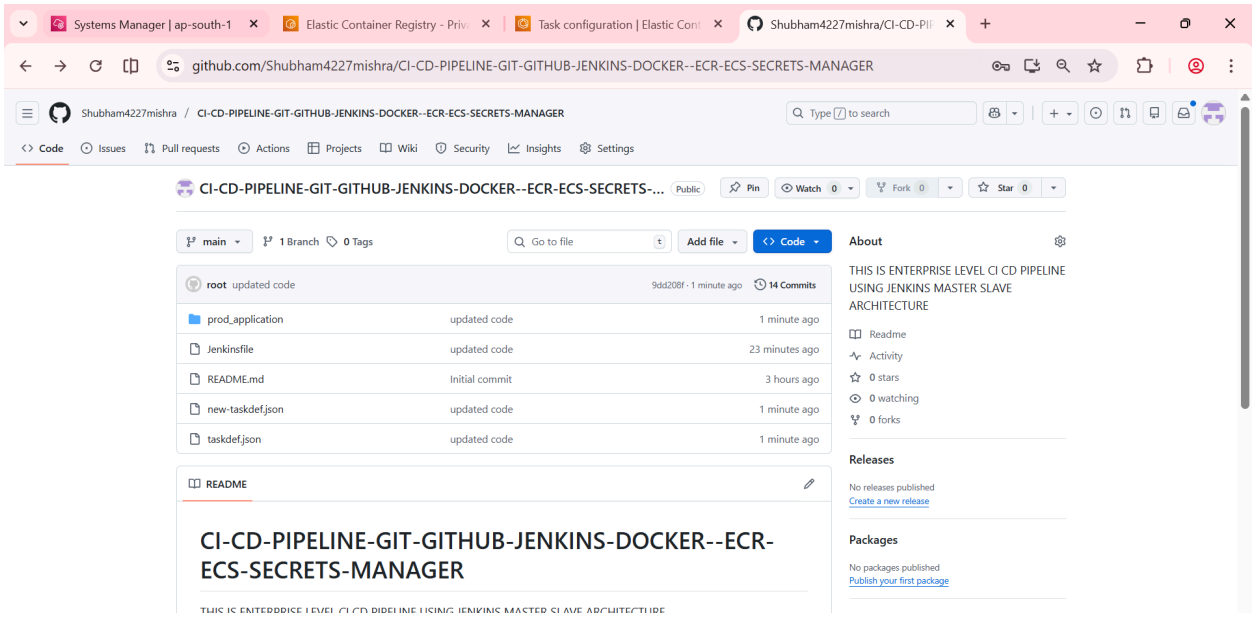
Step 32) Now we will create webhook in github to connect jenkins to github



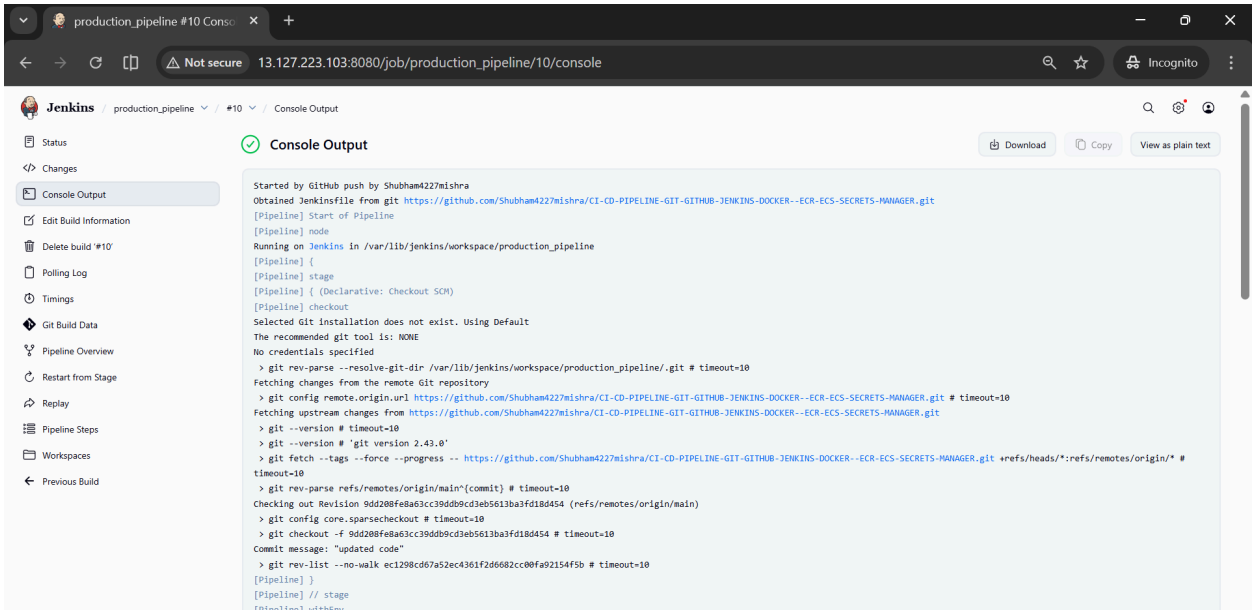
Step 33) Now we will update the code and push it from git to github and then we should see jenkins job automatically triggered



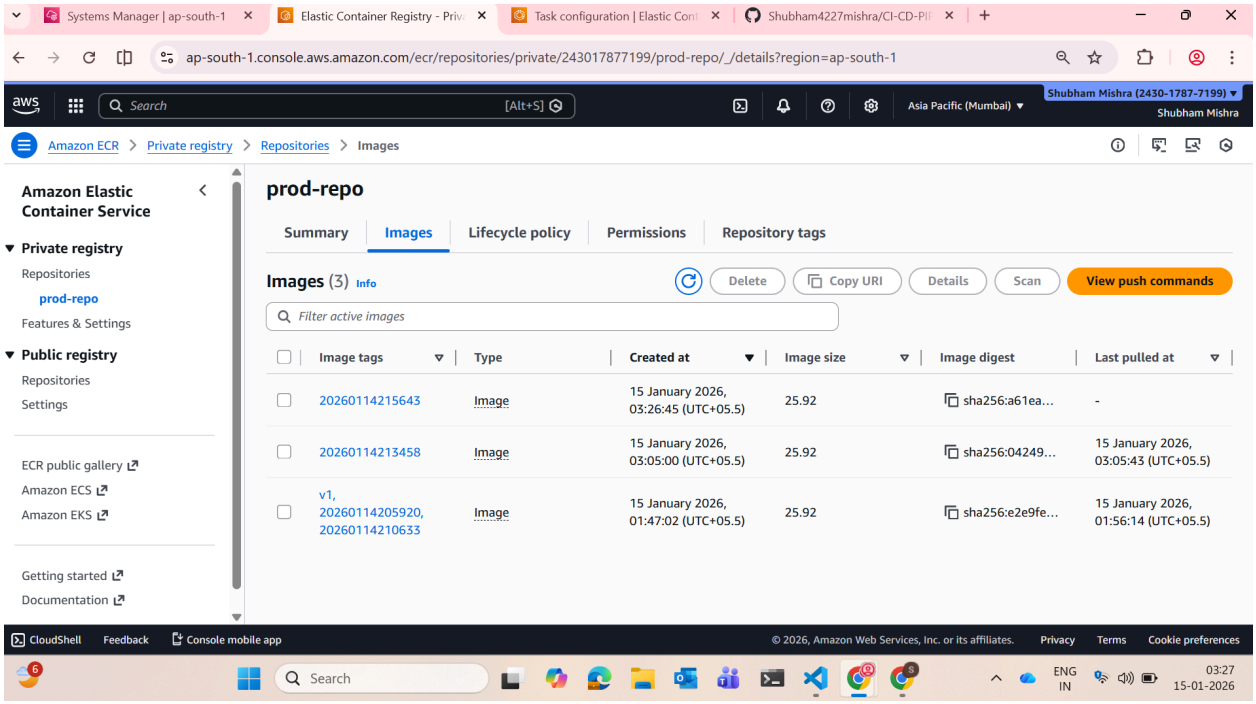
Step 34) As soon as new code pushed we can see it in github



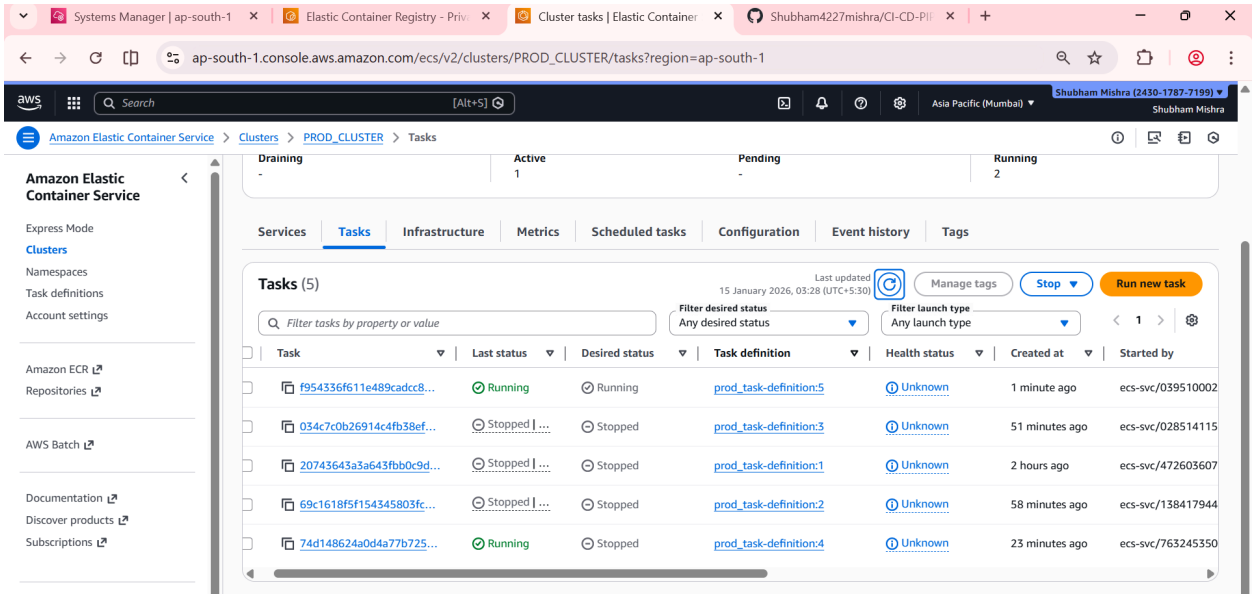
Step 35) Now jenkins detected updated code in github so it started job automatically



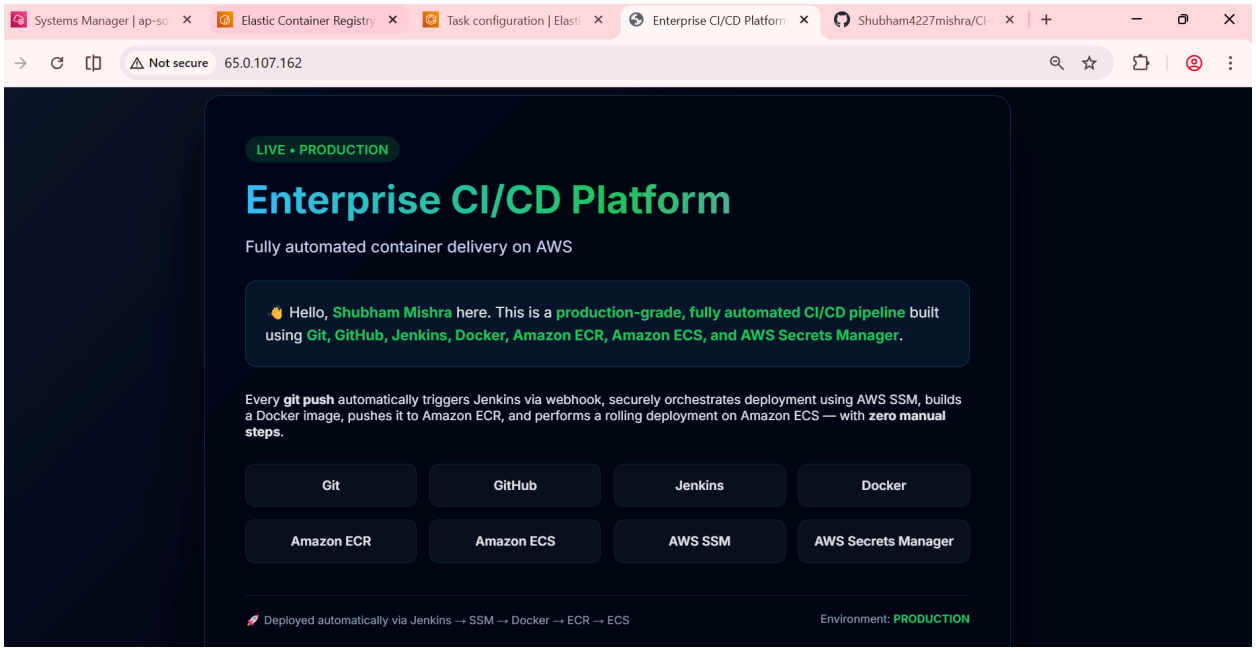
Step 36) New image created automatically



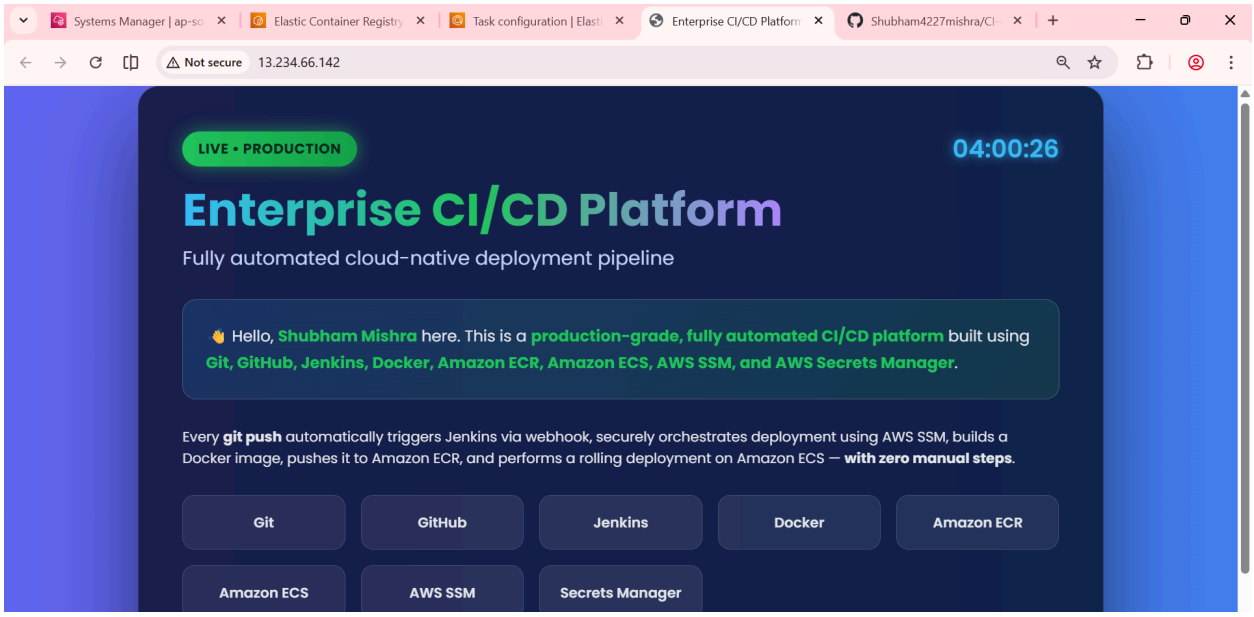
Step 37) New task updated to new image and we can see new container started



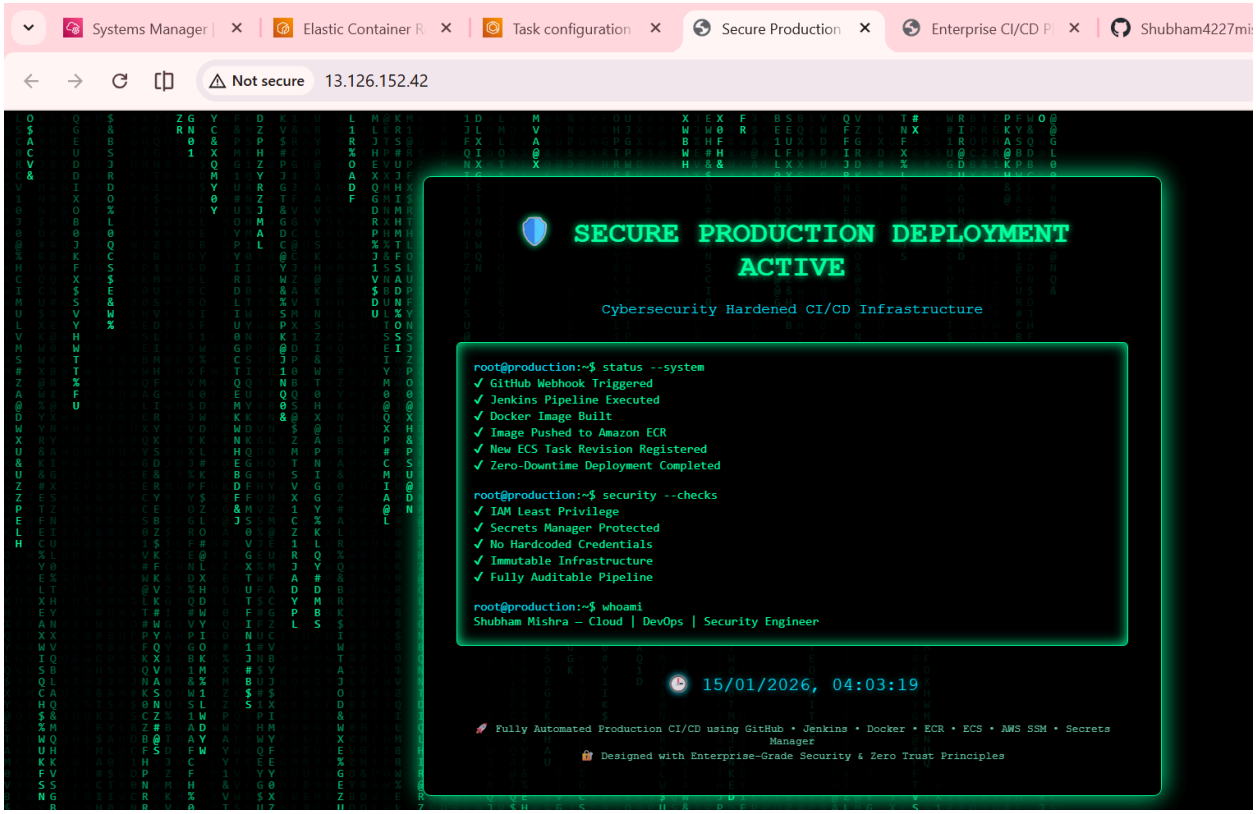
Step 38) Now we can the updated website on public ip of task

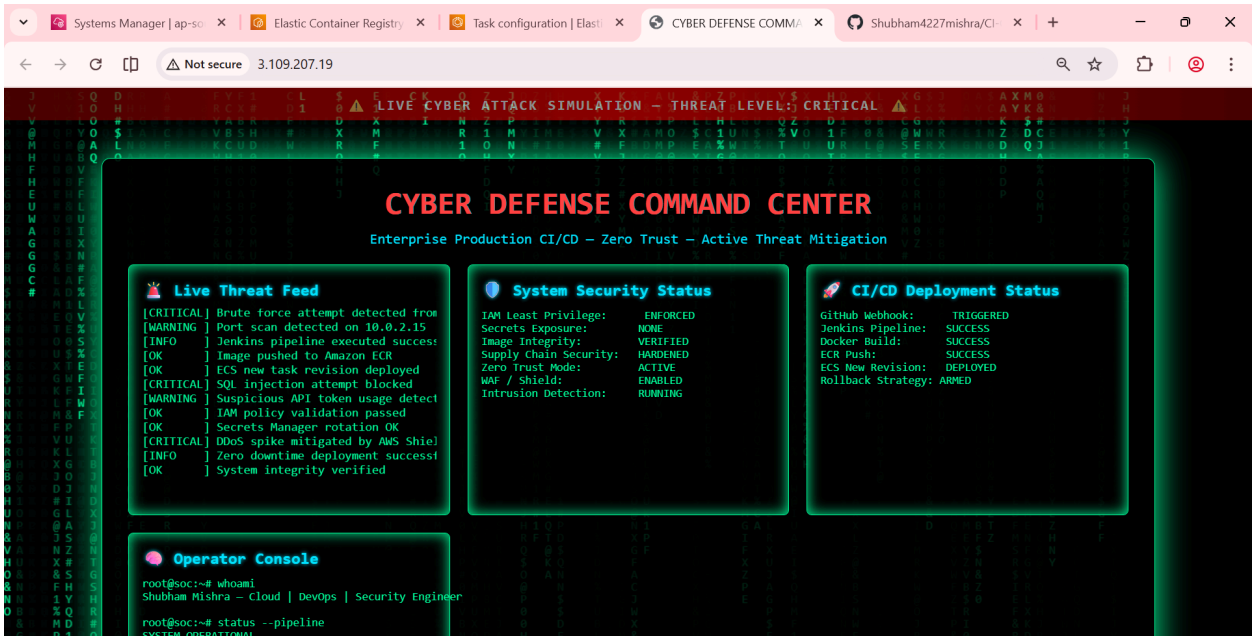


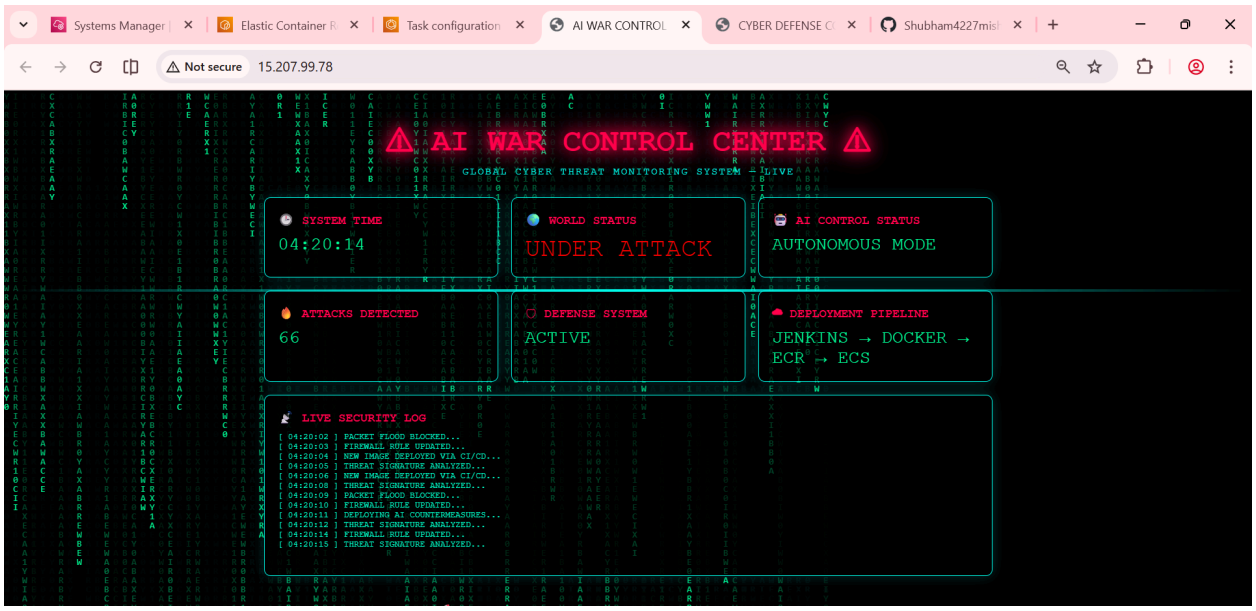
Step 39) Similarly again pushing new code we can see new website updated one



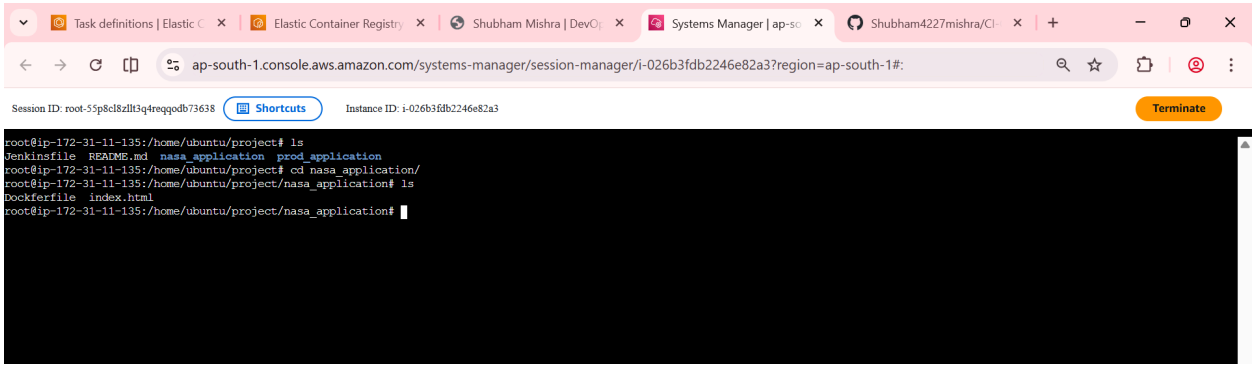
Step 40) Again pushed new code new website

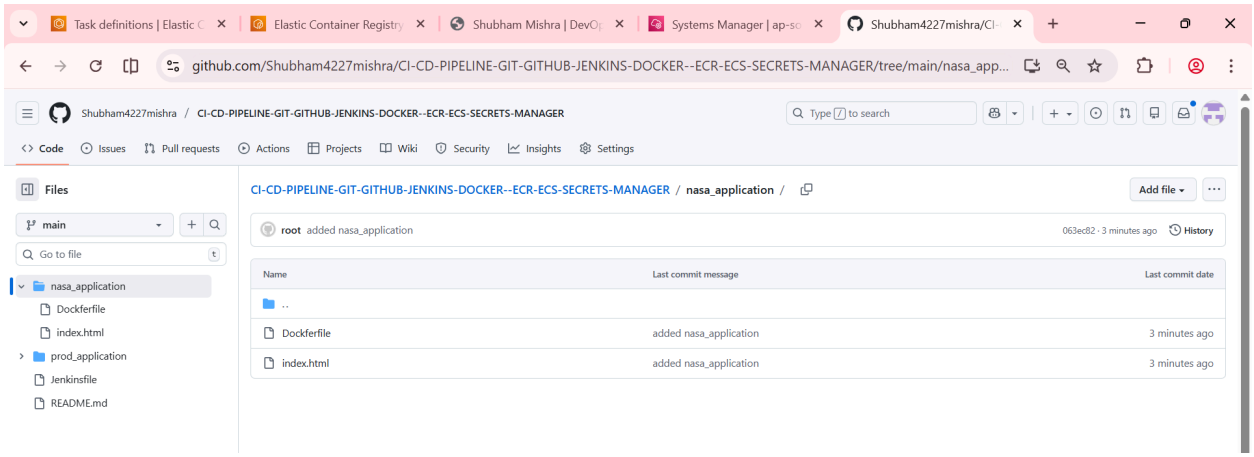




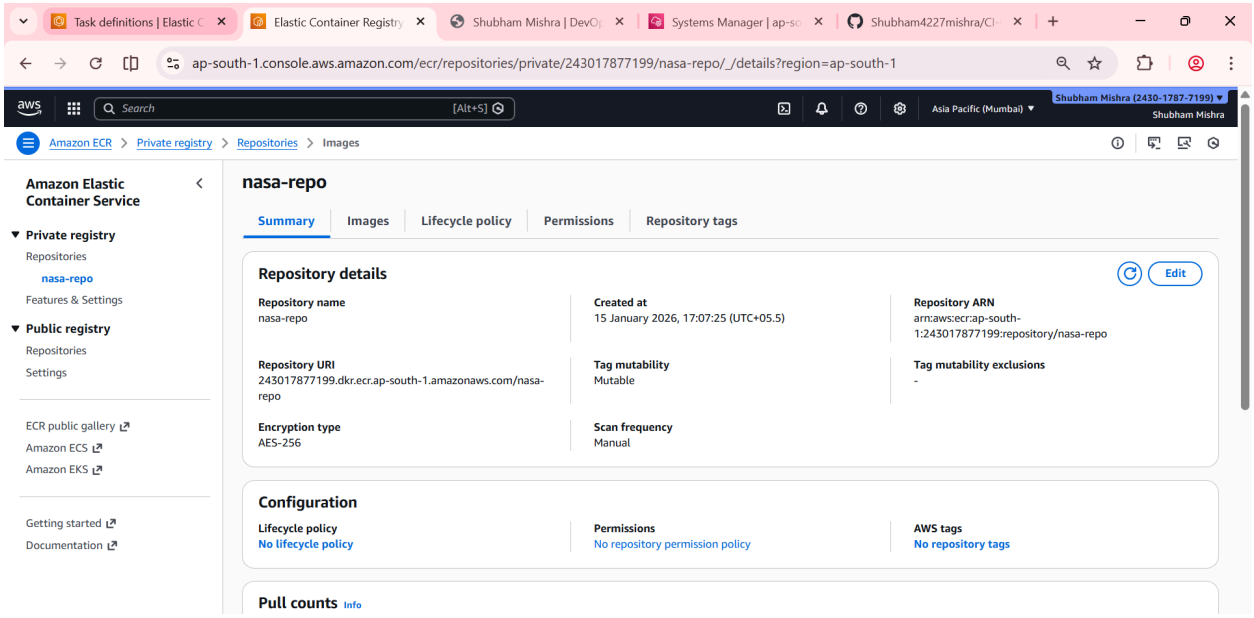


Step 41) Now we will add one more project in git and then push it into github





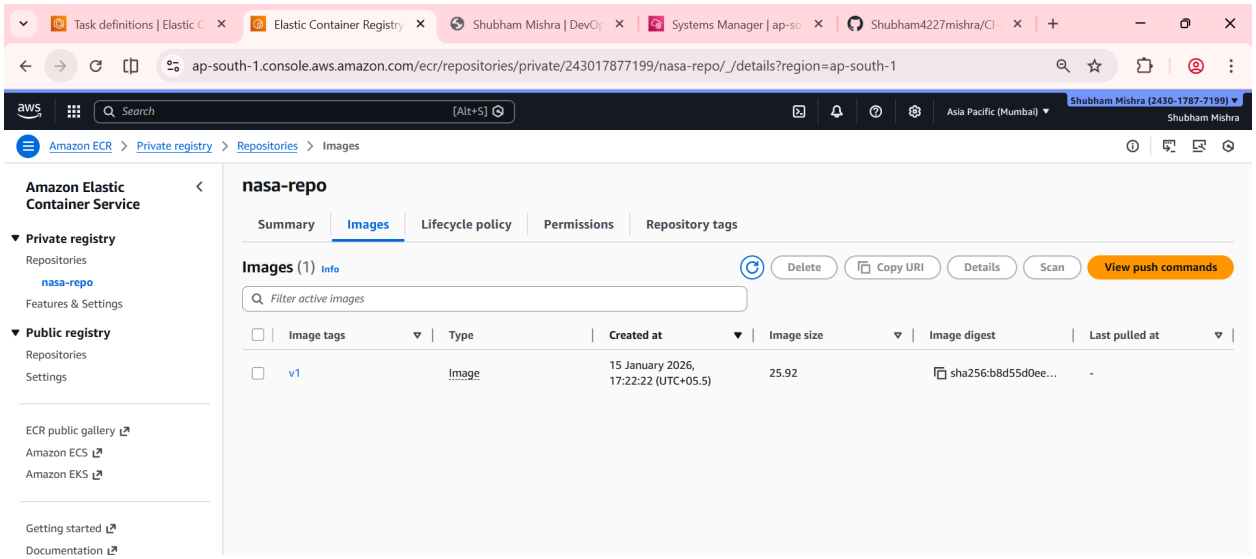
Step 42) Now we will create one more ECR for nasa_application



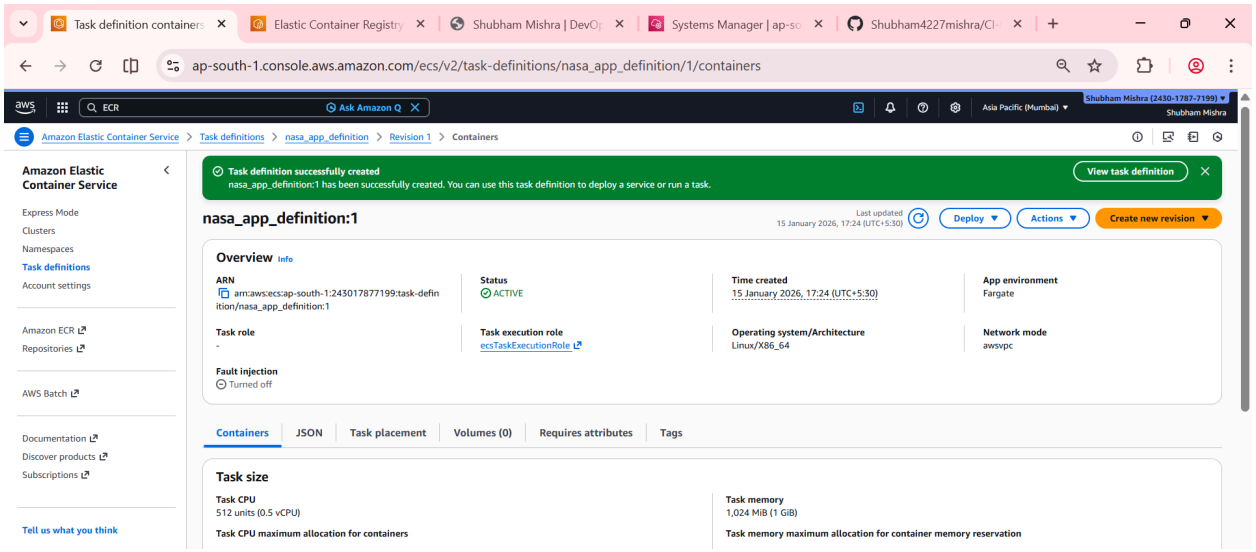
Step 43) We will login into ecr nasa-repo

```
Run 'docker build --help' for more information
root@ip-172-31-11-135:/home/ubuntu/project/nasa_application# aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin 243017877199.dkr.ecr.ap-south-1.amazonaws.com

WARNING! Your credentials are stored unencrypted in '/root/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/

Login Succeeded
root@ip-172-31-11-135:/home/ubuntu/project/nasa_application#
```

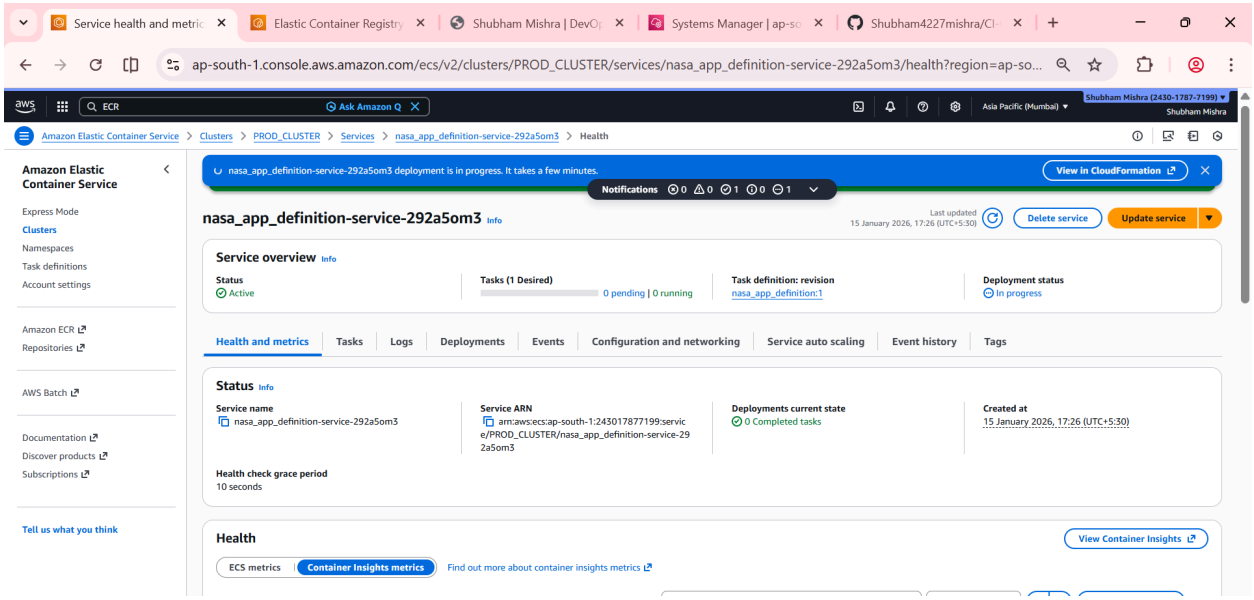
Step 44) Docker image pusehd in ECR for nasa_application



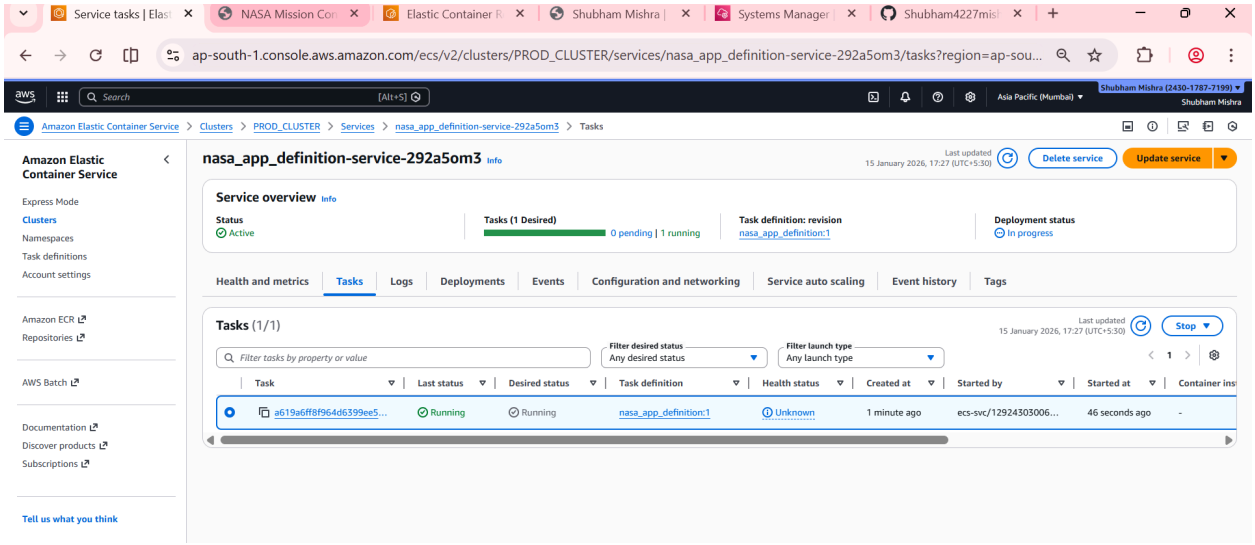
Step 45) Task definition for nasa_application



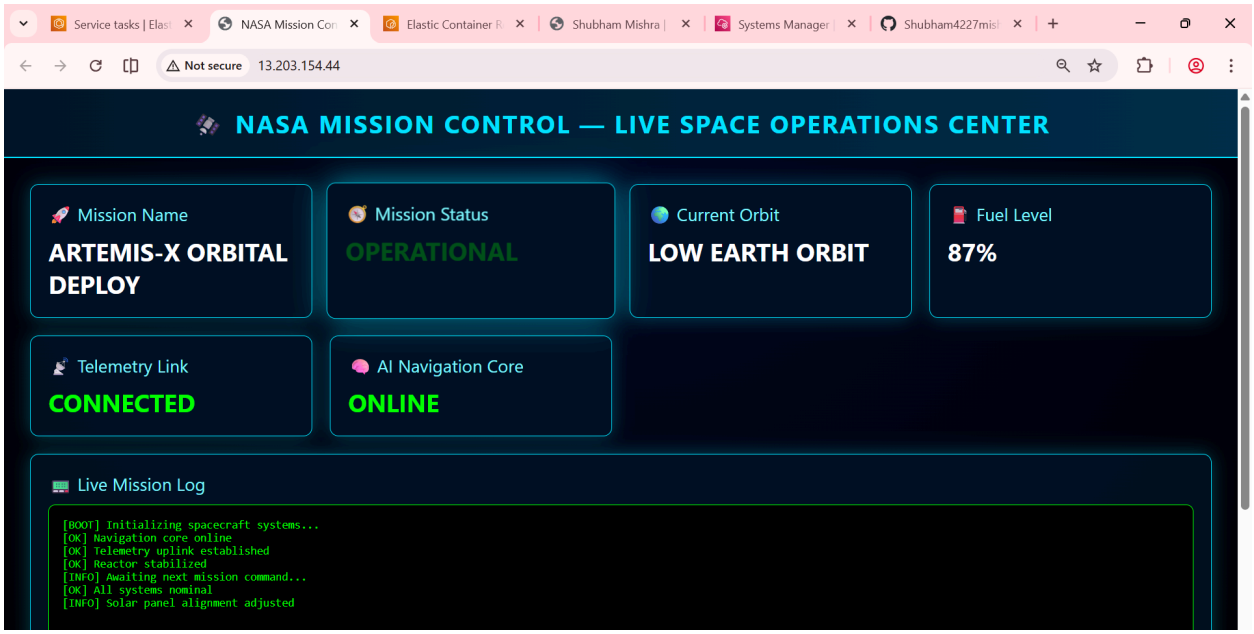
Step 46) Created new service for nasa_application to use nasa_application task definition



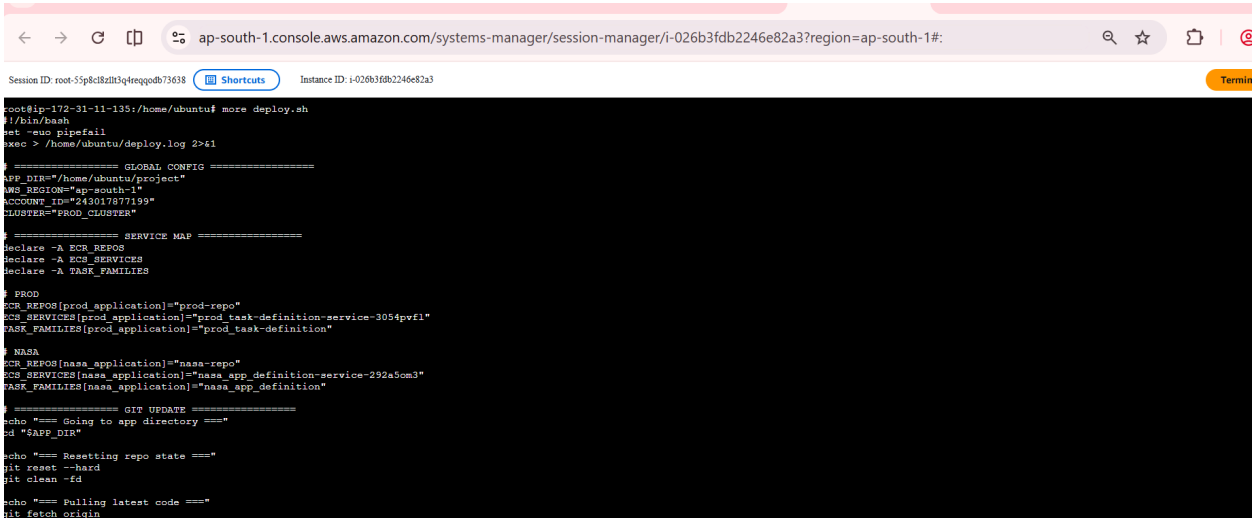
Step 47) Now new task container created successfully for nasa_application

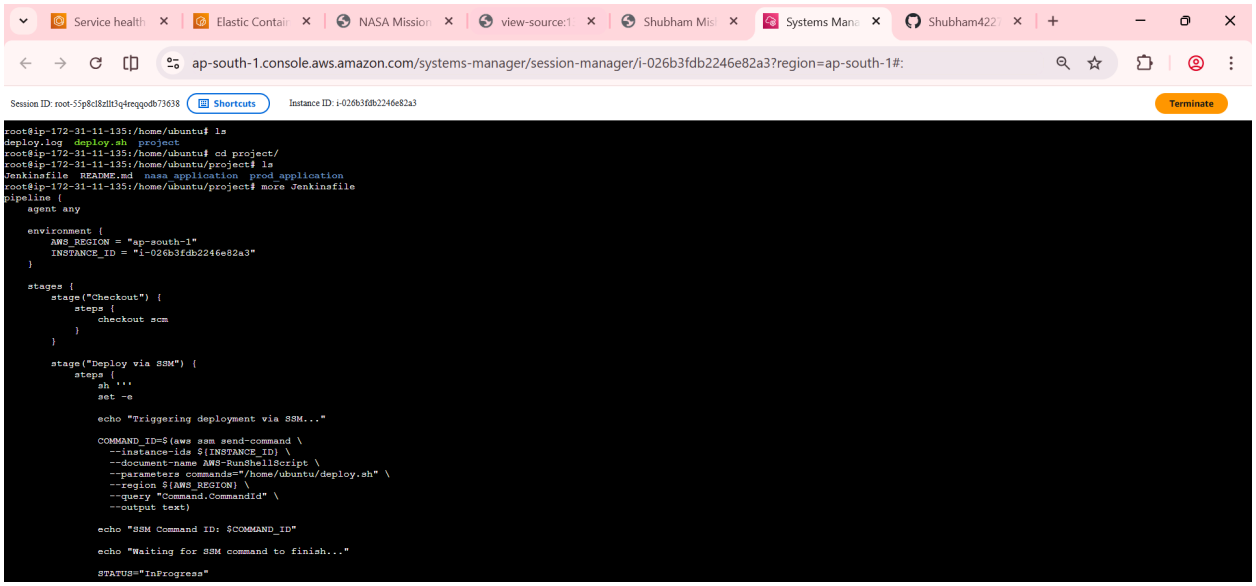


Step 48) We can verify the nasa_application website

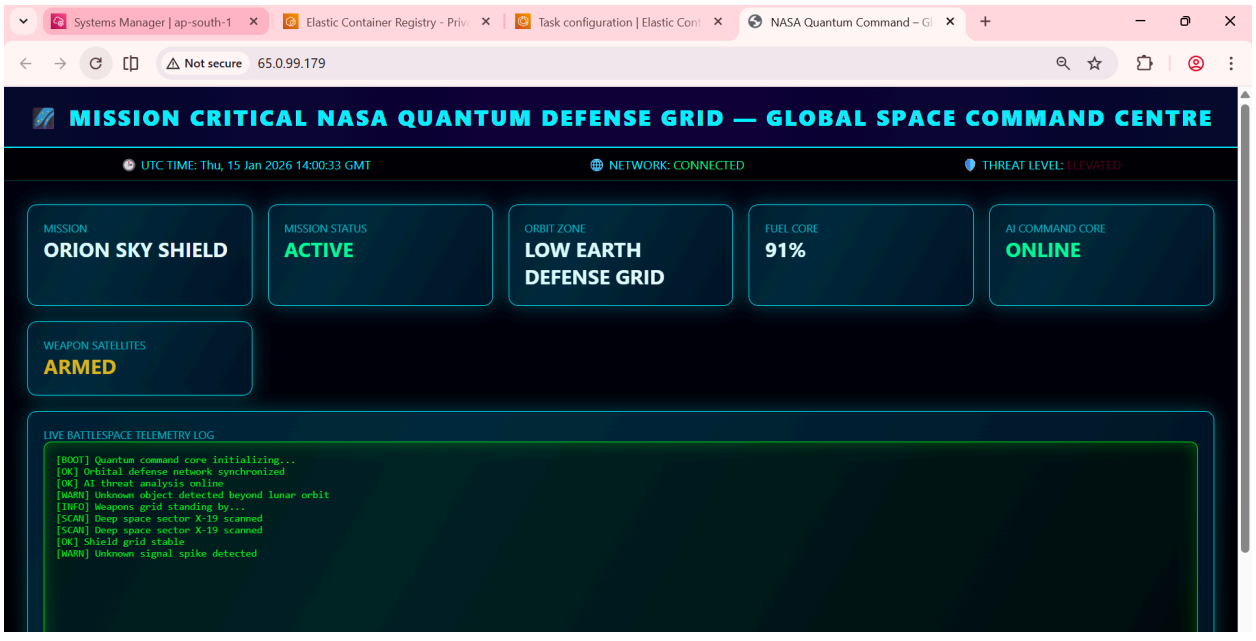


Step 49) Now we will update deploy.sh and Jenkinsfile to fully automate this on new code changes it should build new task

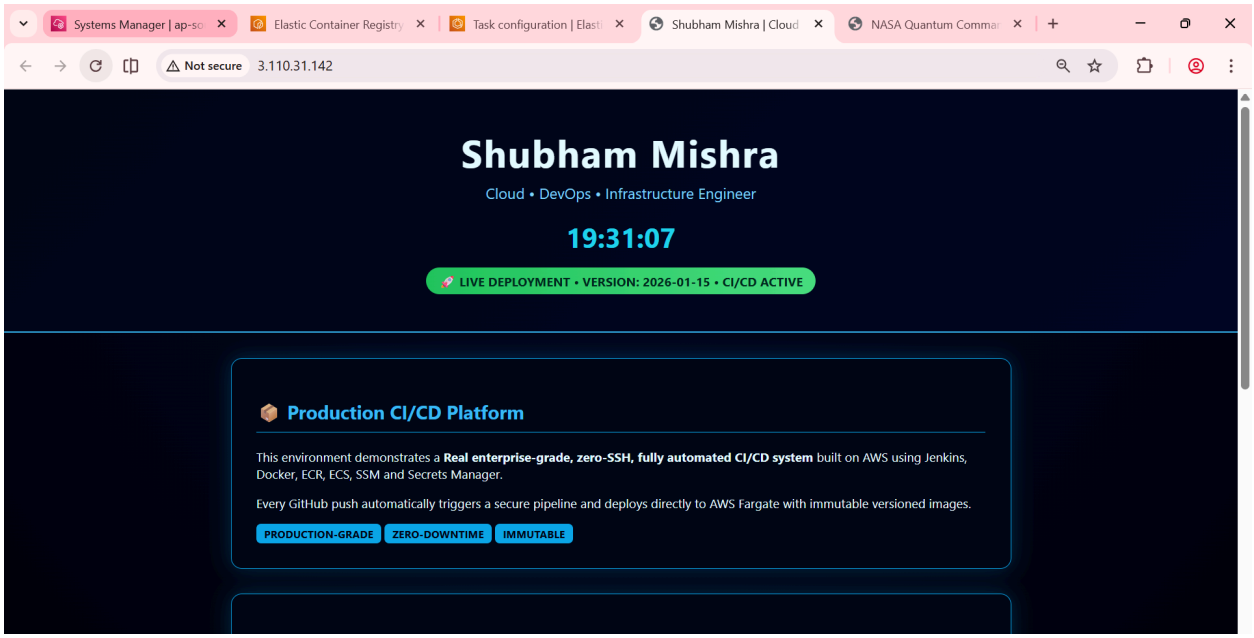




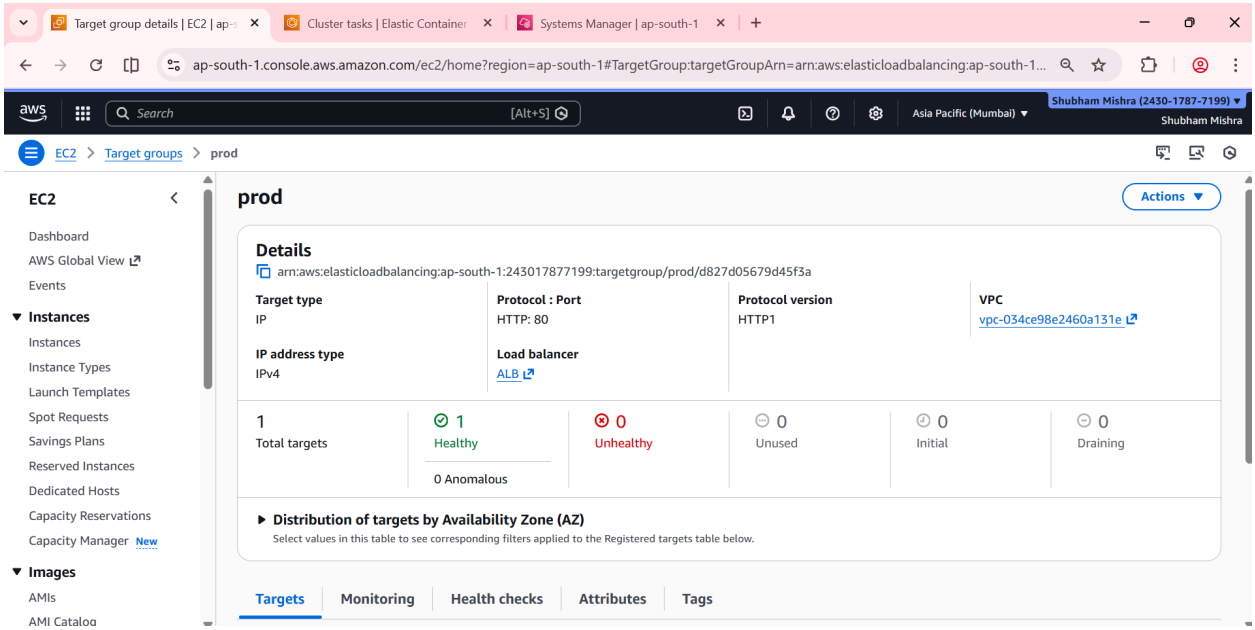
Step 50) Now we will make changes in nasa_application code and check the fully automated pipeline

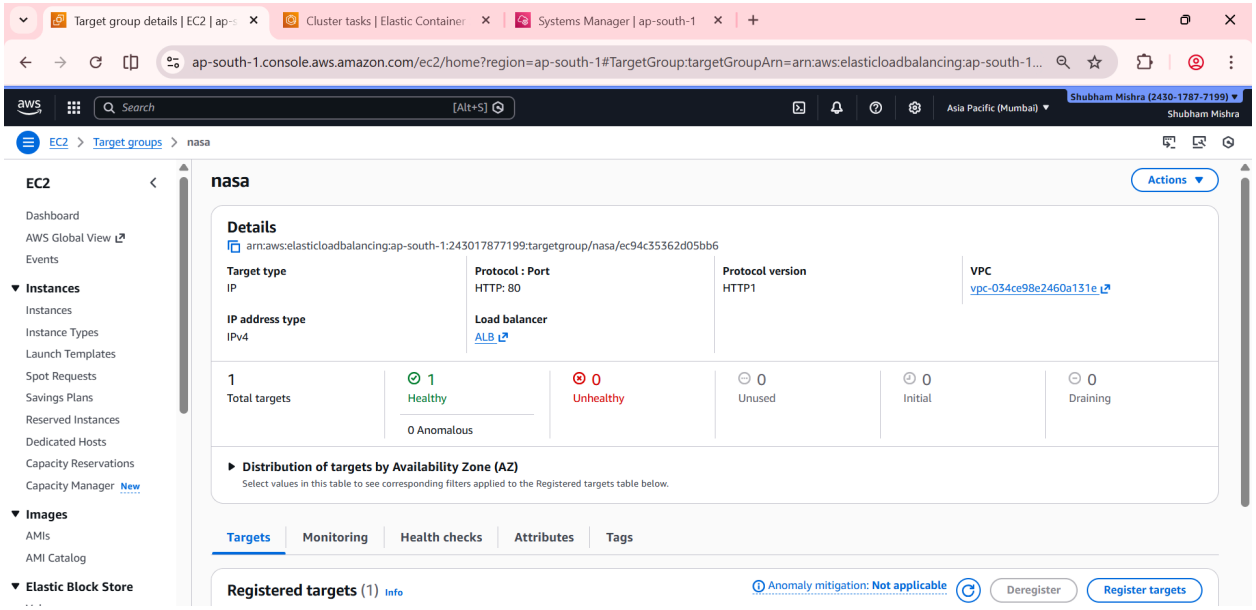


Step 51) Also making change in prod application also we can see the automateed deployment

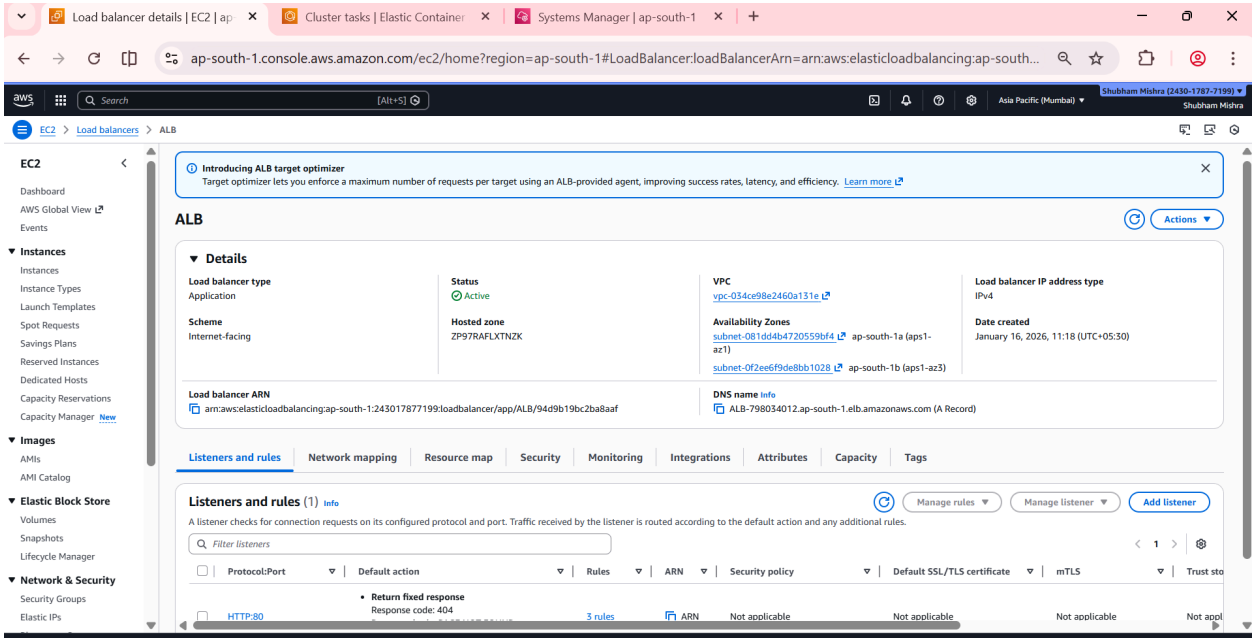


Step 52) Now we will create target group for nasa and prod application

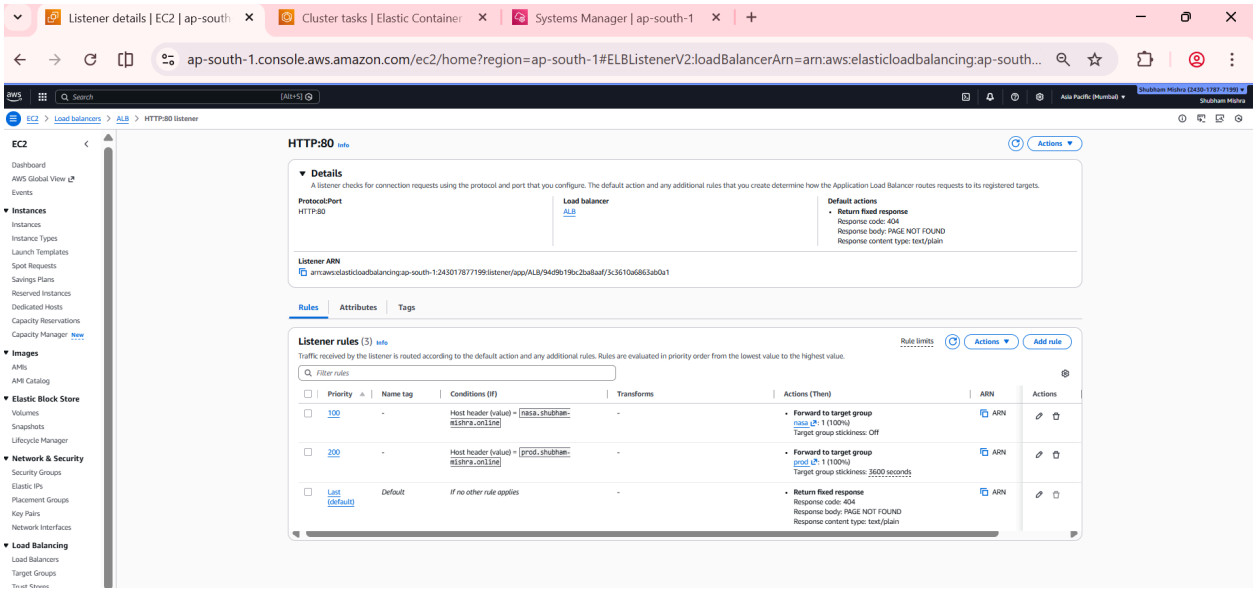




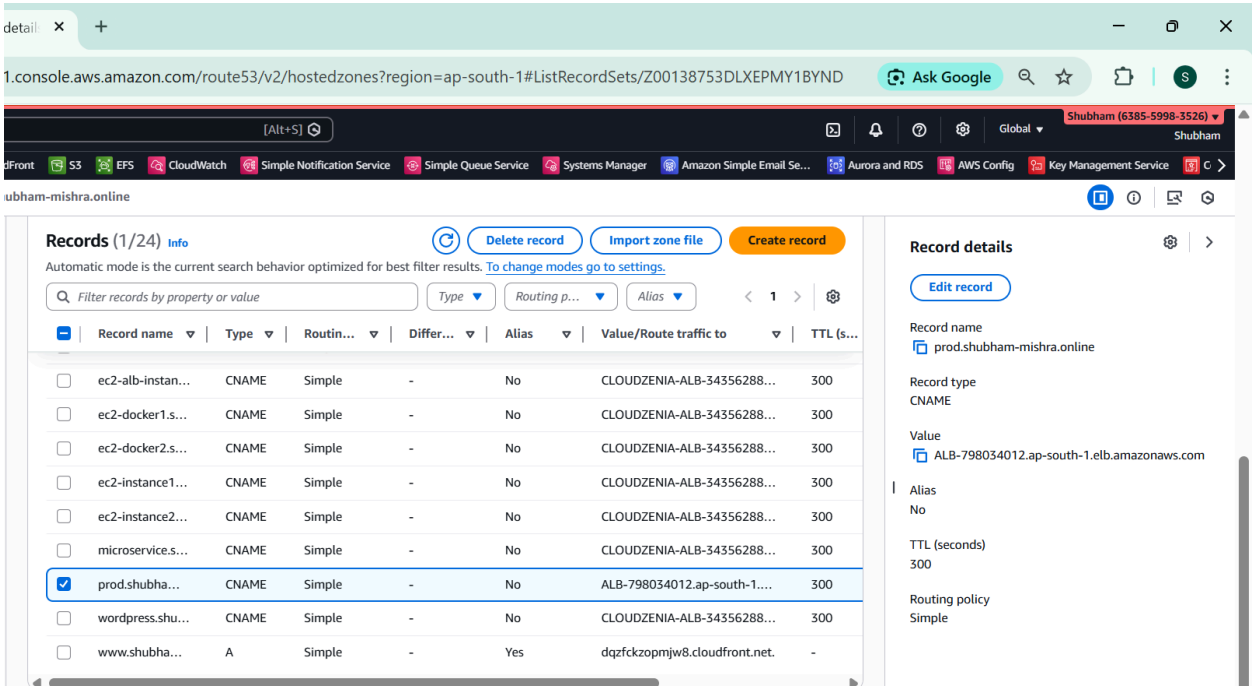
Step 53) Now we will create Application Load Balancer and add this target group

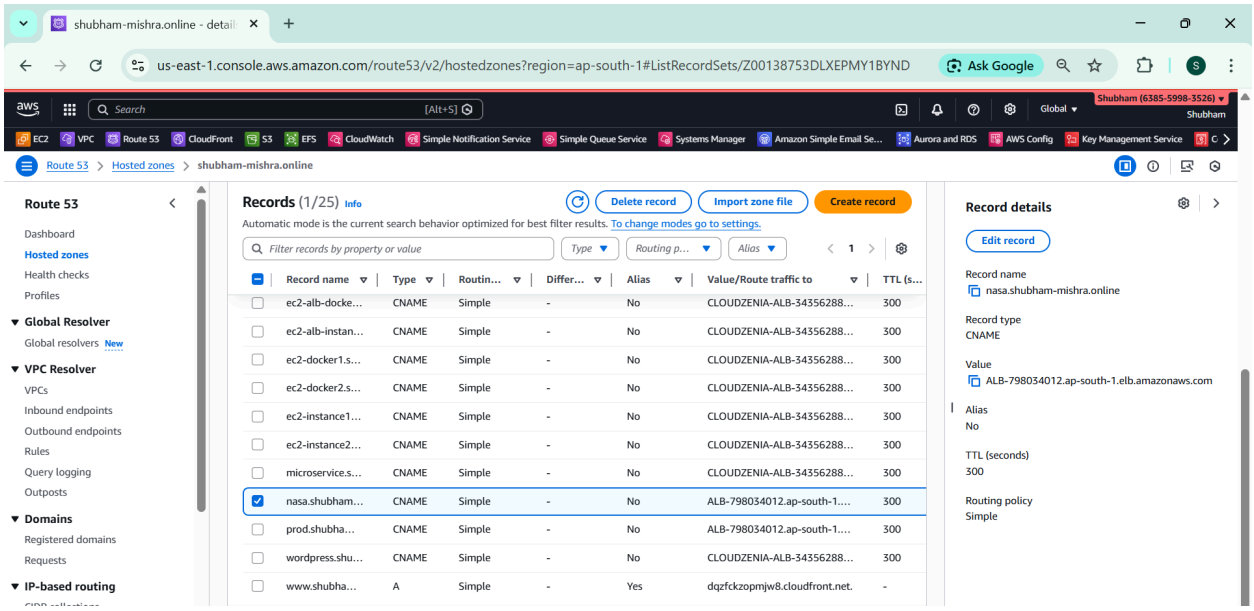


Step 54) Now we will create rules on port 80 for Application load balancer bases on host header based routing

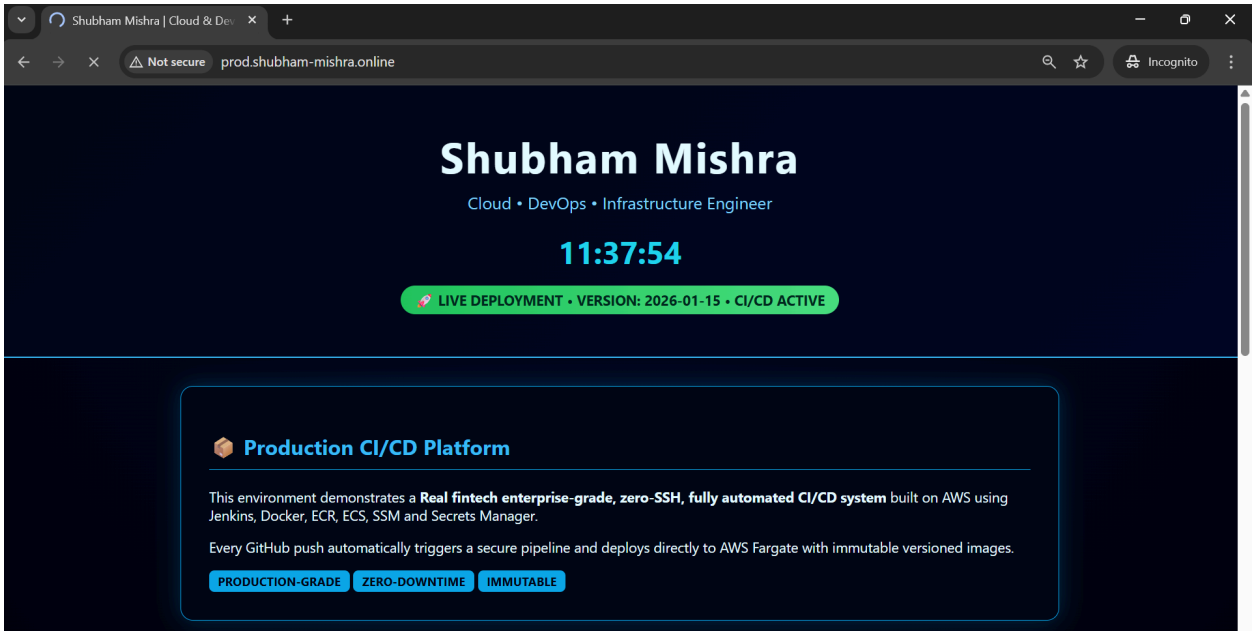


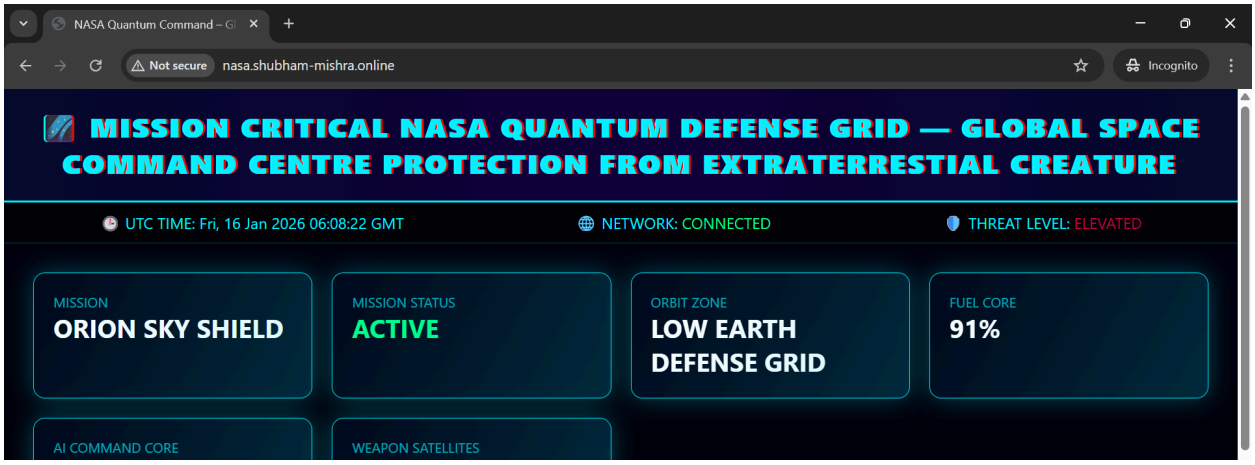
Step 55) Now we will add this ALB as CNAME record in route53 public hosted zone in other aws account so to route traffic based on the host header



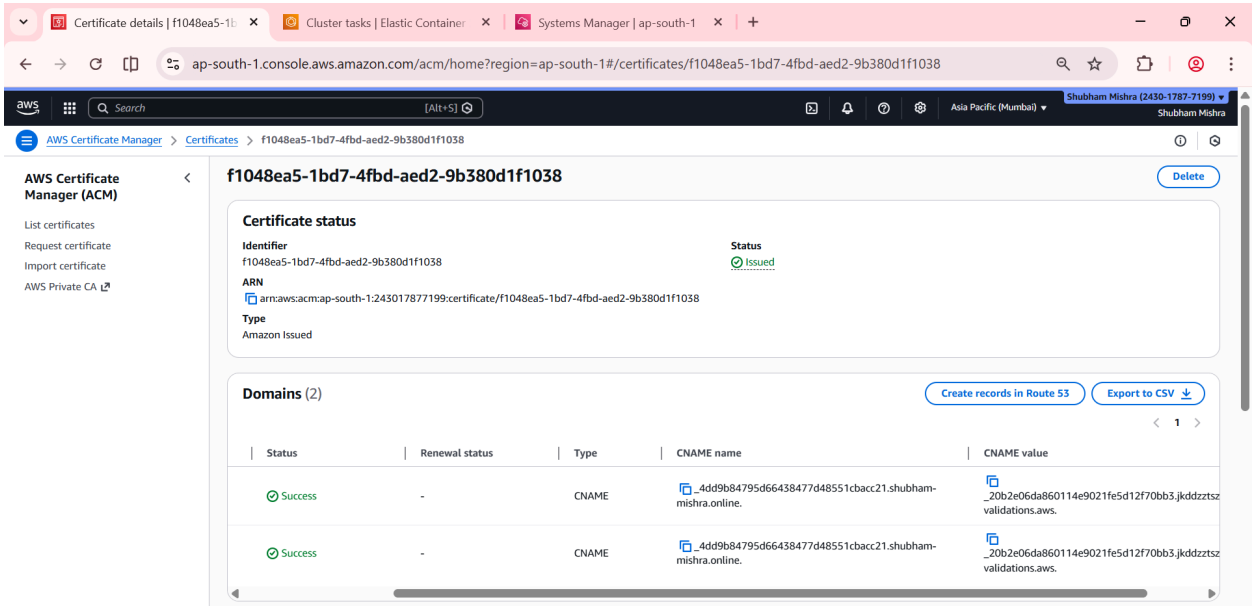


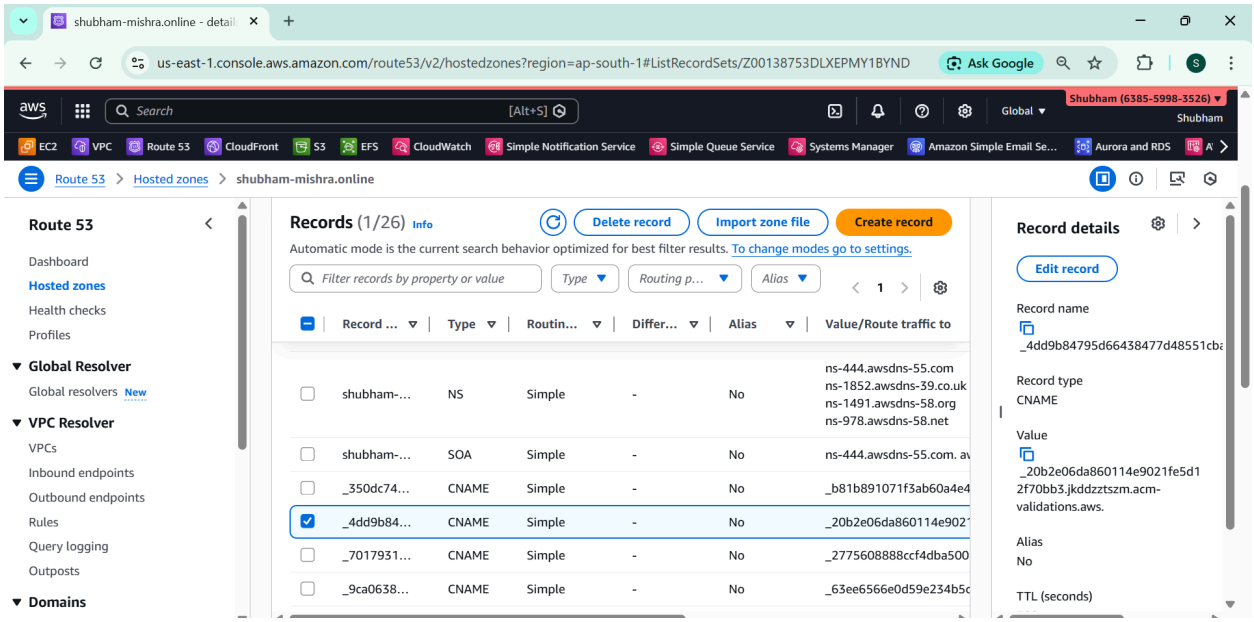
Step 56) Now we will verify the setup based on host header it should got to particular ecs task



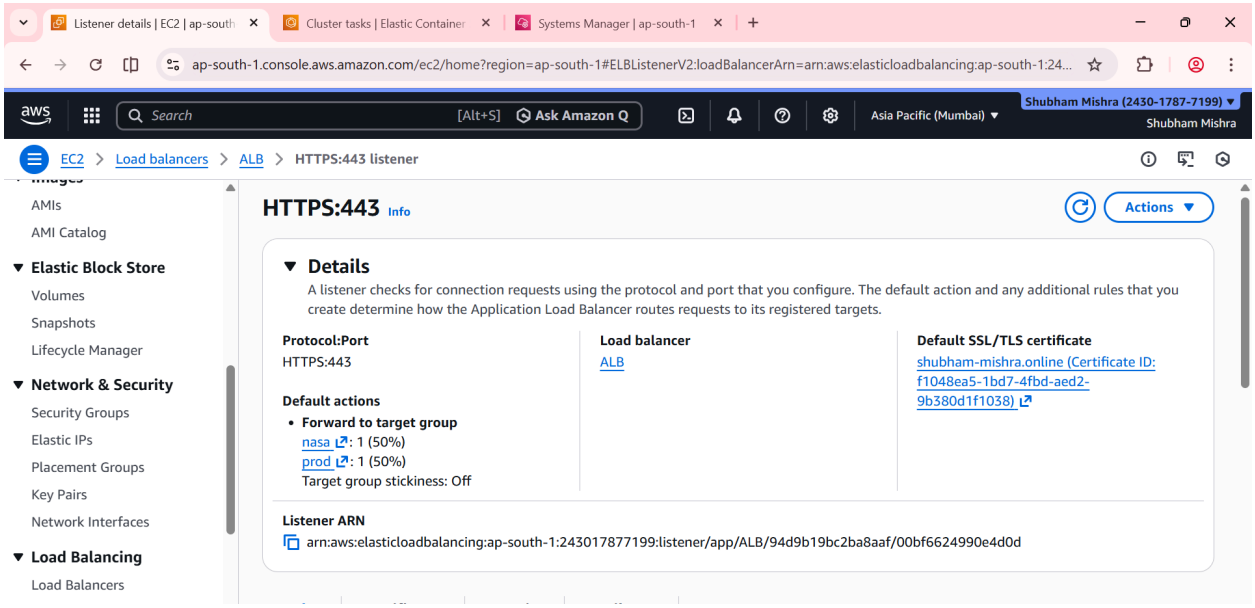


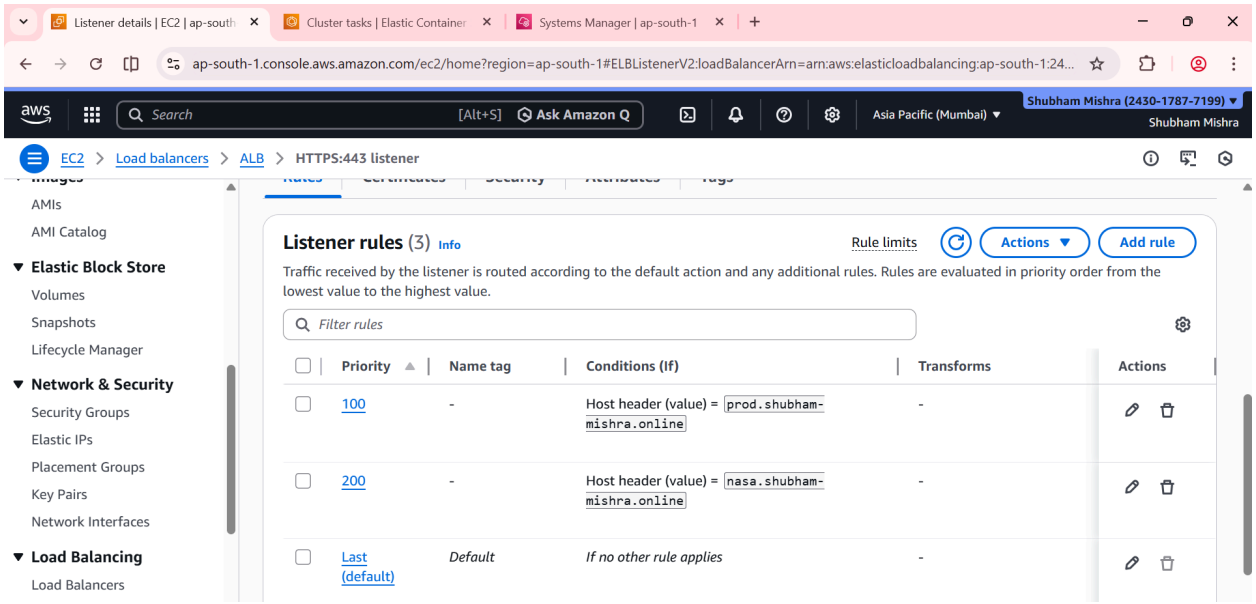
Step 57) Now we will create SSL certificate and add the records in route53 to use this certificate



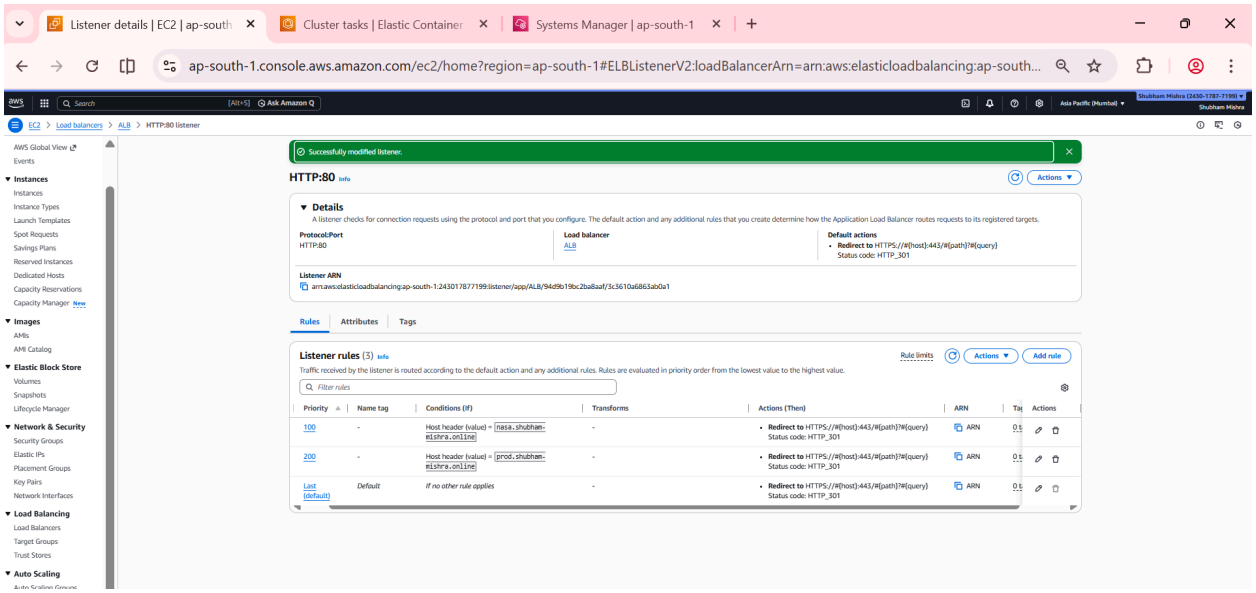


Step 58) Now we will add https 443 listener and add this ssl certificate over there

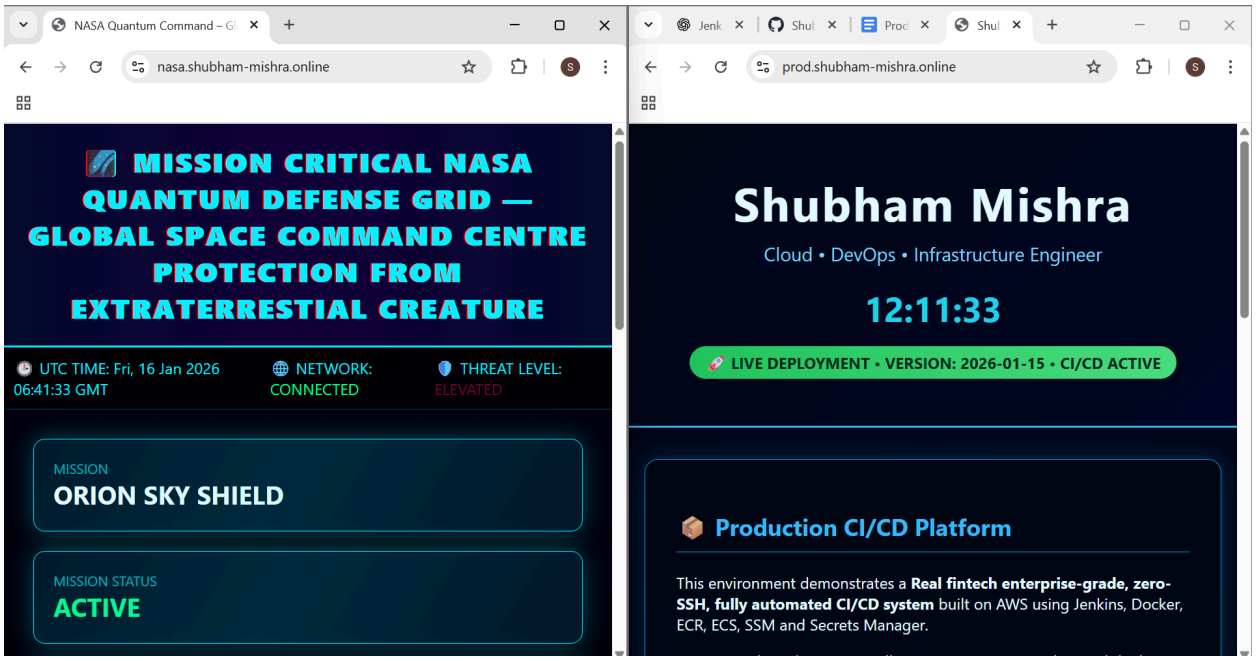




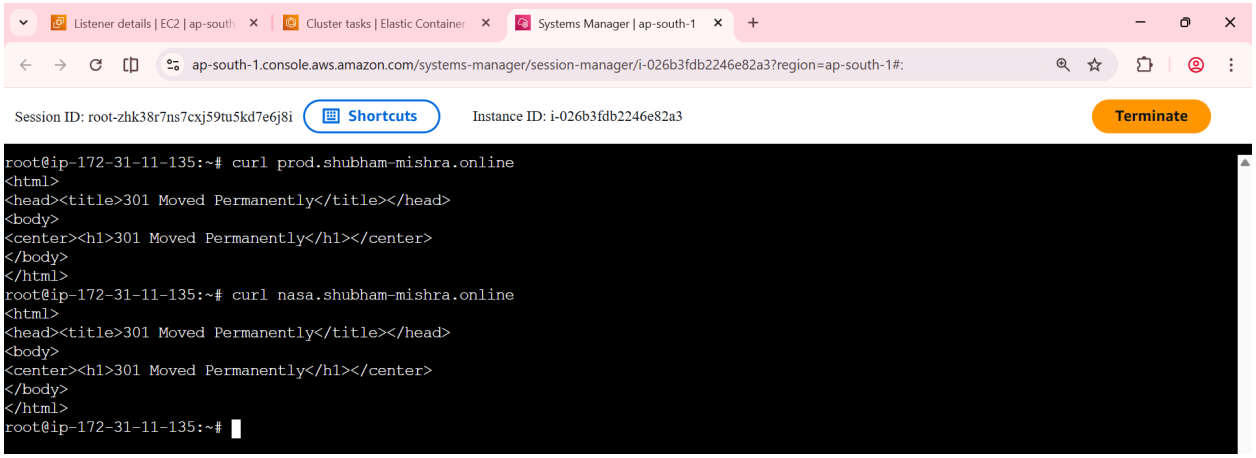
Step 59) Now we will also redirect http request to https

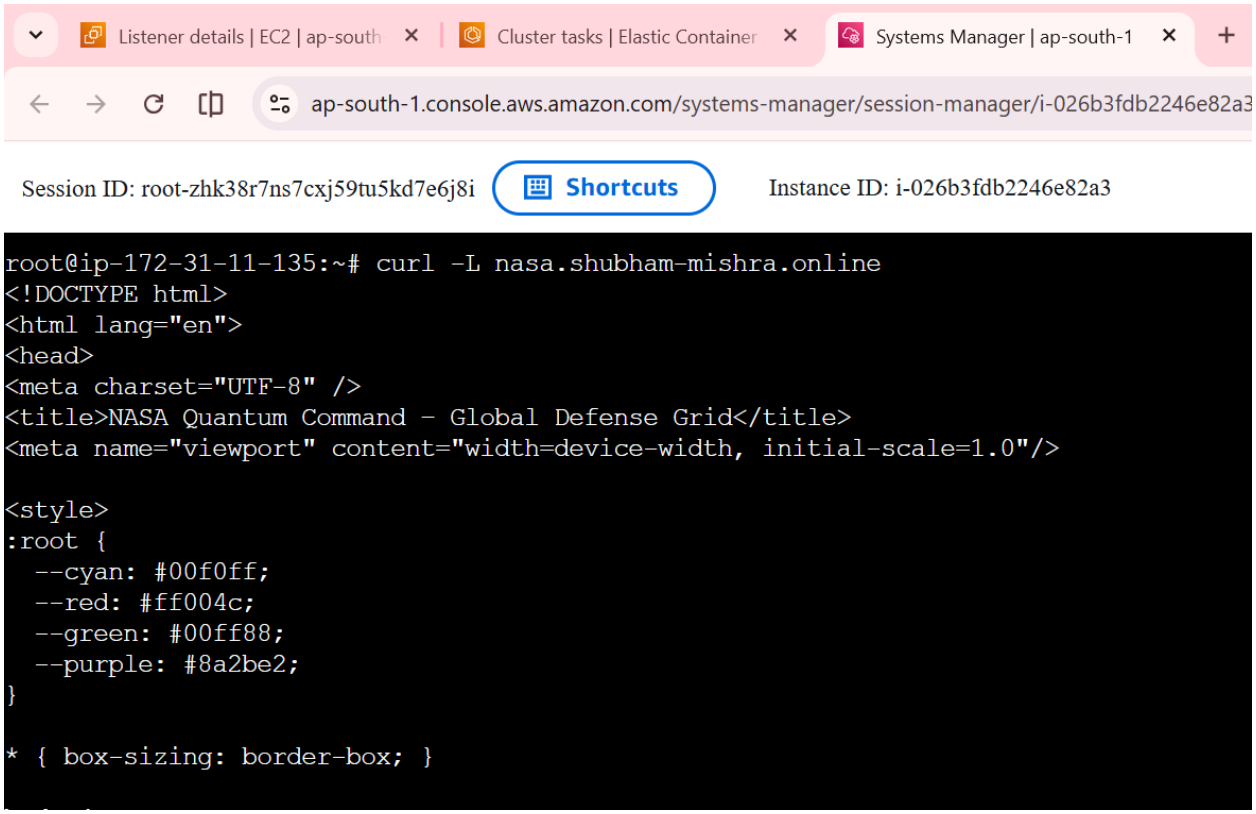


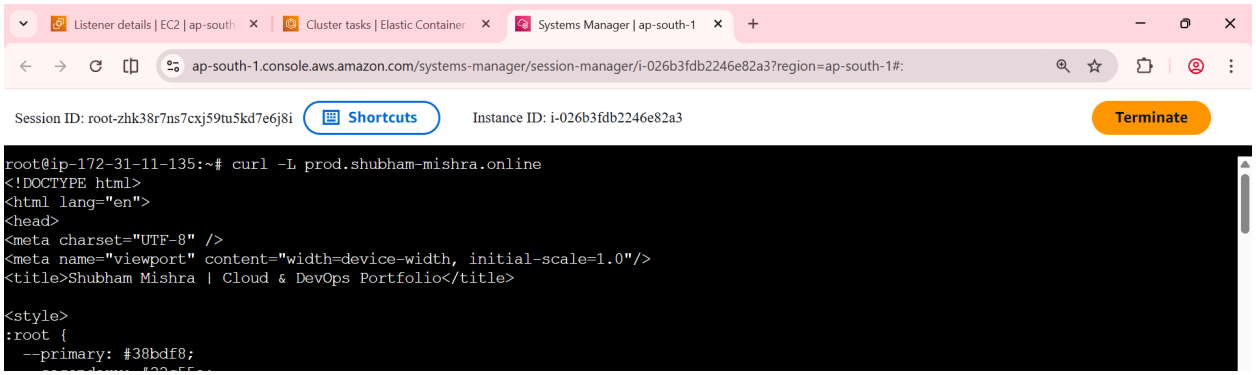
Step 60) Now we will test whether our ssl setup is working corectly and also http request is getting redirected to https



Also see the http request is not redirected to https

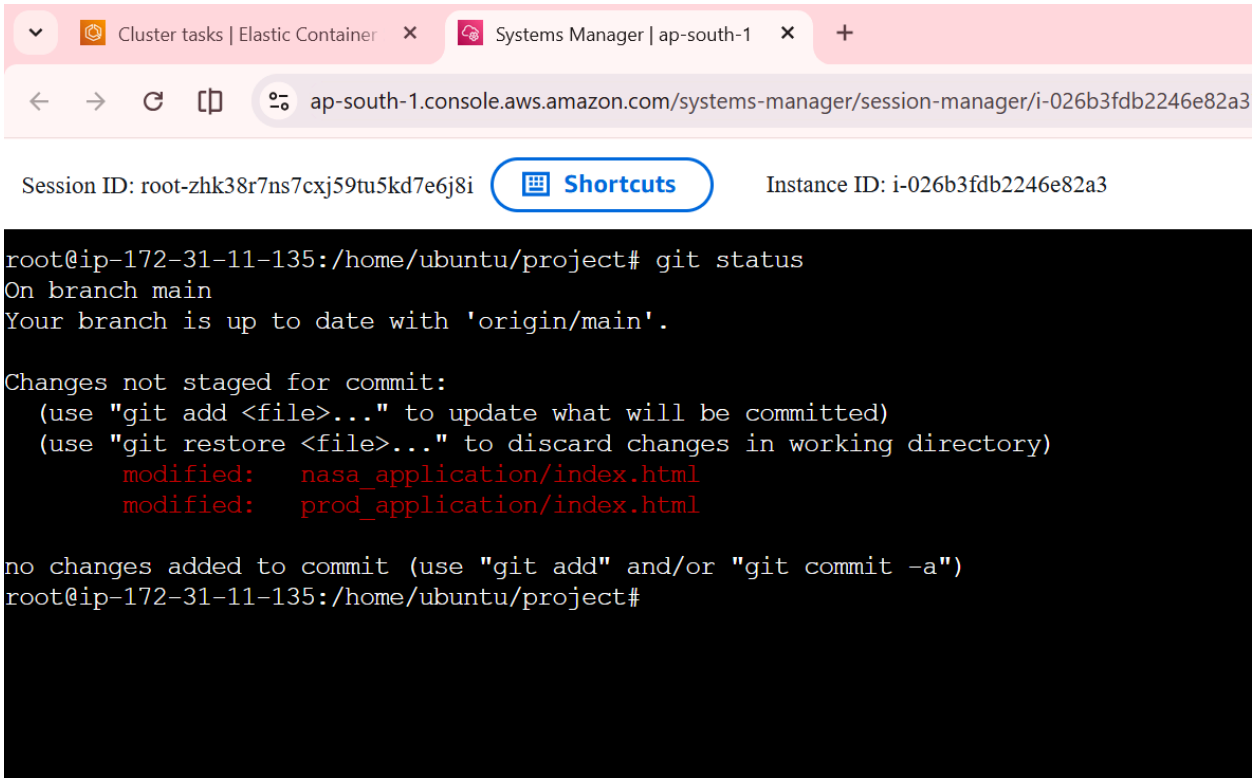




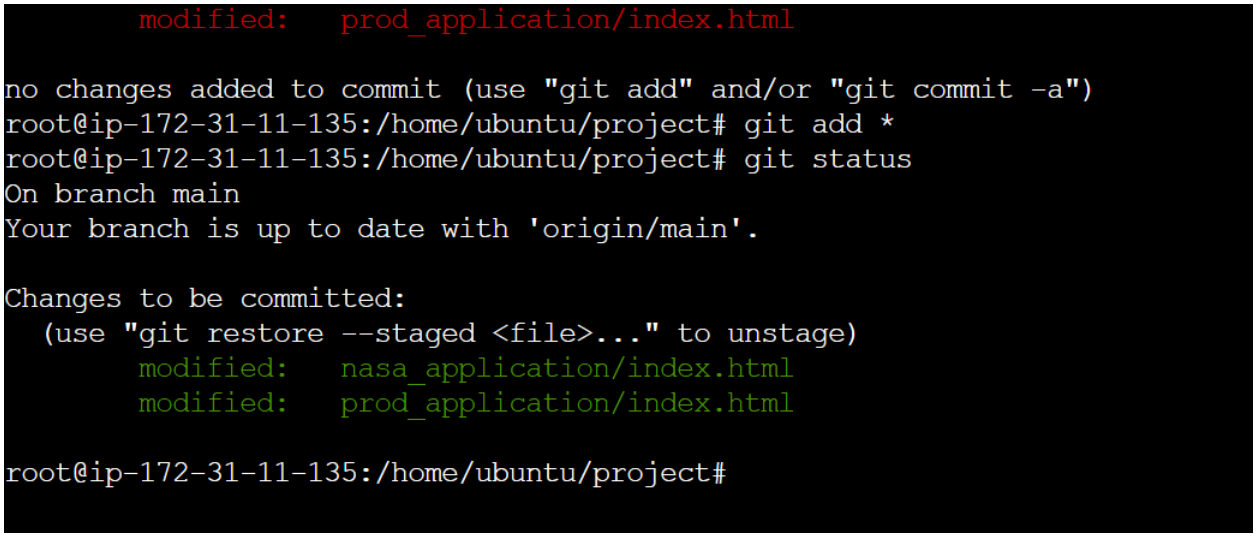


Step 61) Now last final step to check when new code pushed to the github then those changes are reflected into our website and user is seeing updated website

a) We can see changes have been made in the code for both nasa and prod application



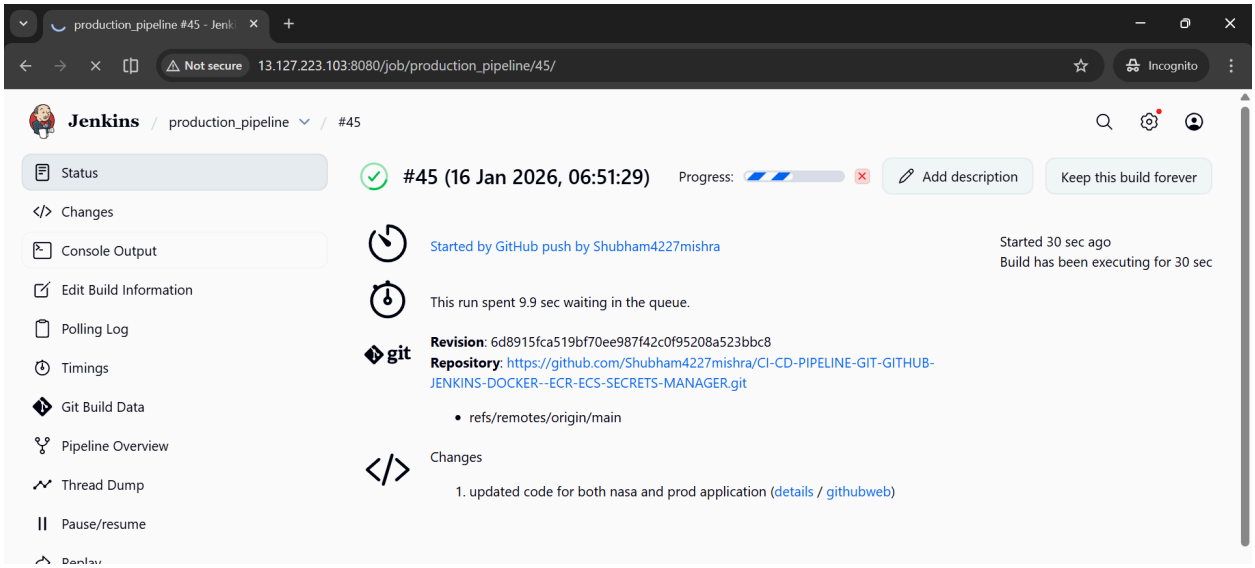
b) Now we will commit and push this changes in github



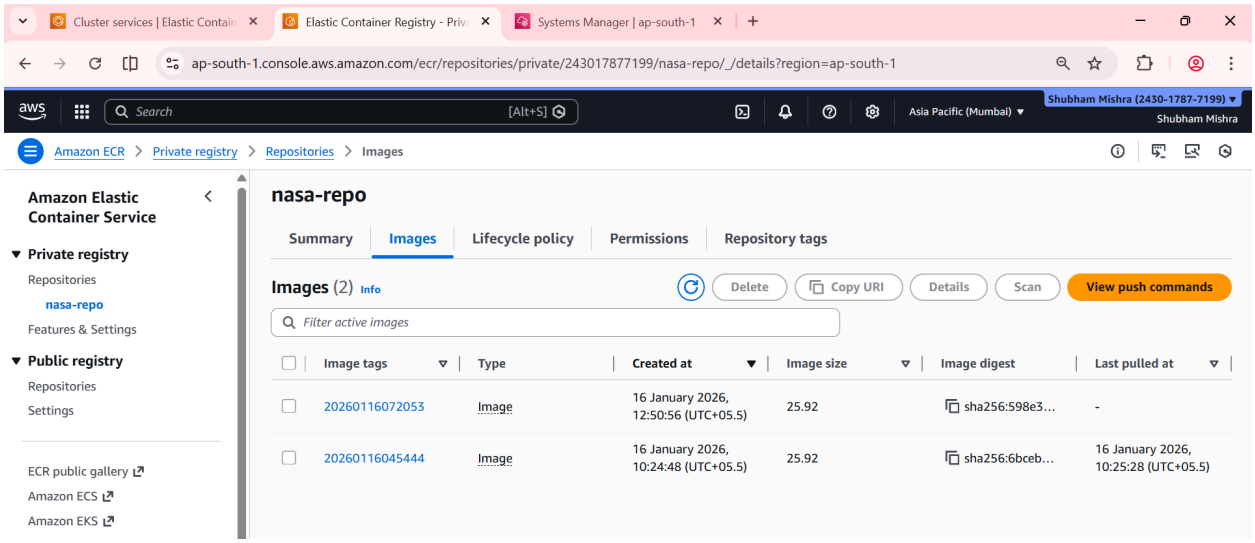
```
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   nasa_application/index.html
        modified:   prod_application/index.html

root@ip-172-31-11-135:/home/ubuntu/project# git commit -m "updated code for both nasa and prod application"
[main 6d8915f] updated code for both nasa and prod application
 Committer: root <root@ip-172-31-11-135.ap-south-1.compute.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 2 files changed, 154 insertions(+), 158 deletions(-)
```

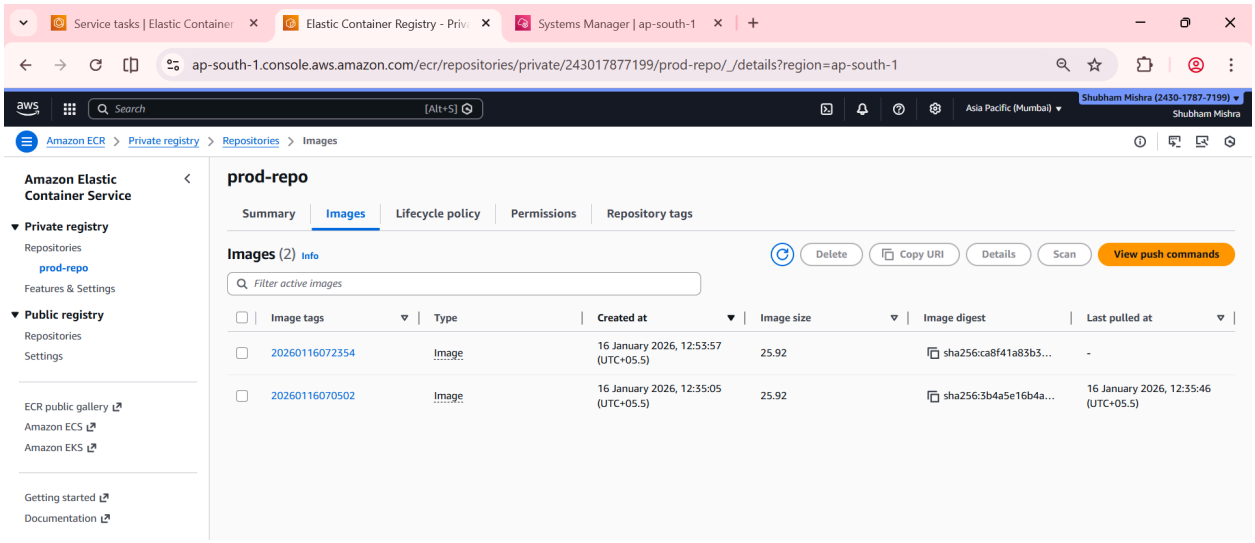
```
root@ip-172-31-11-135:/home/ubuntu/project# git commit -m "updated code for both nasa and prod application"
[main 6d8915f] updated code for both nasa and prod application
 Committer: root <root@ip-172-31-11-135.ap-south-1.compute.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 2 files changed, 154 insertions(+), 158 deletions(-)
root@ip-172-31-11-135:/home/ubuntu/project# git push origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 3.48 KiB | 3.48 MiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/Shubham4227mishra/CI-CD-PIPELINE-GIT-GITHUB-JENKINS-DOCKER--ECR-ECS-SECRETS-MANAGER.git
   2b2bdef..6d8915f  main -> main
root@ip-172-31-11-135:/home/ubuntu/project#
```

c) Now jenkins job started running

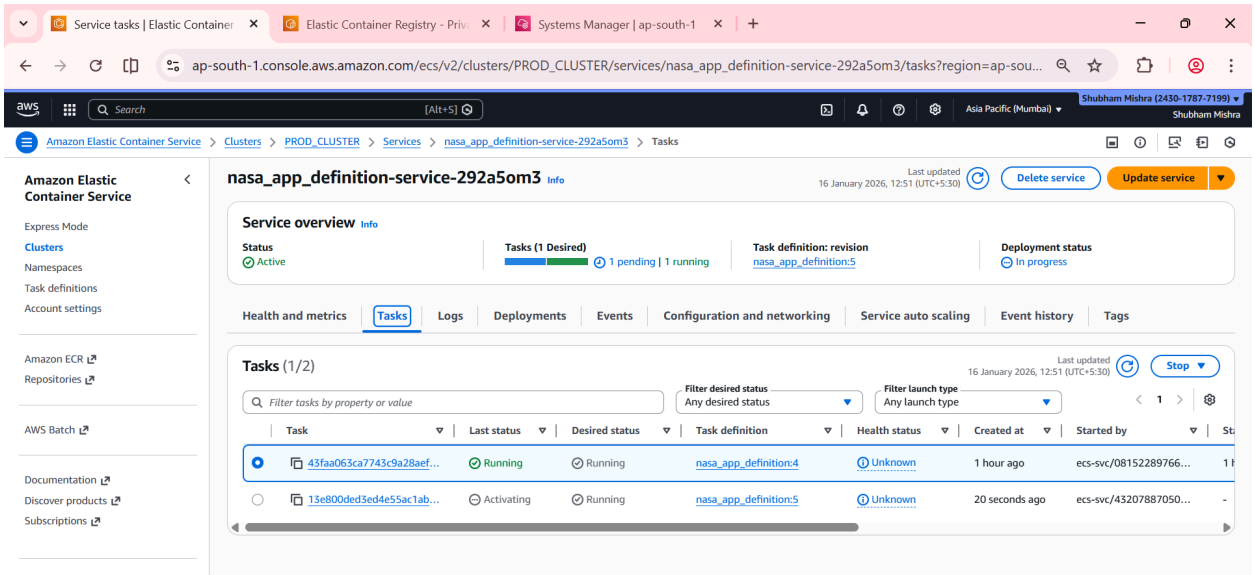


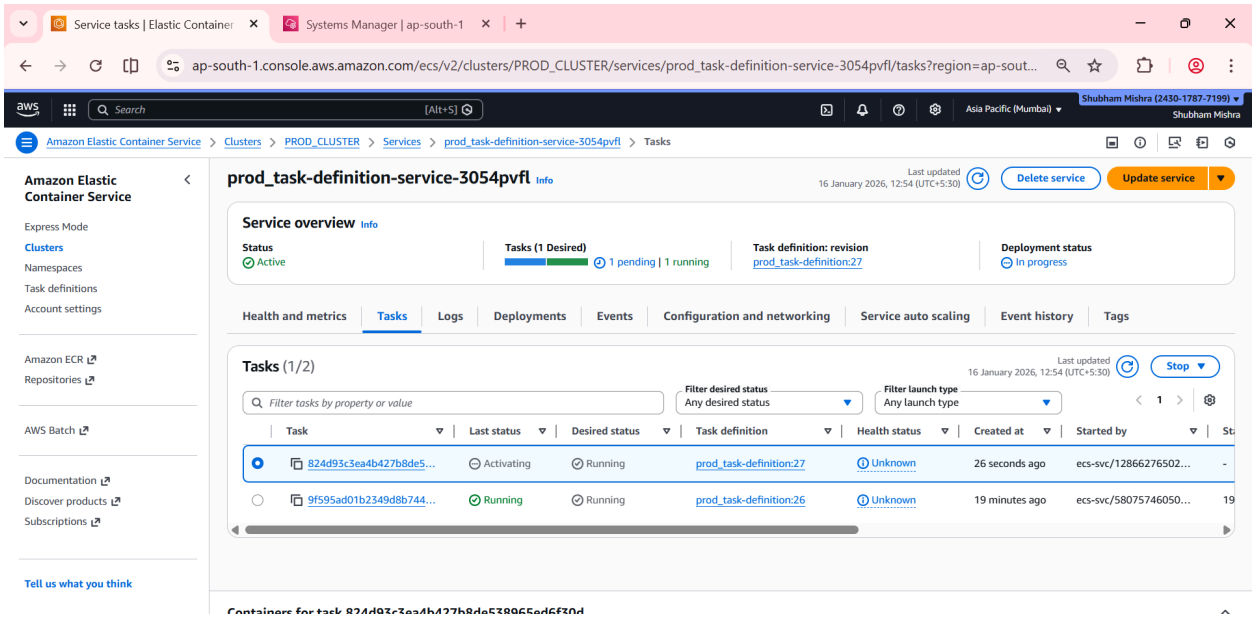
d) Now we can see new images in ECR



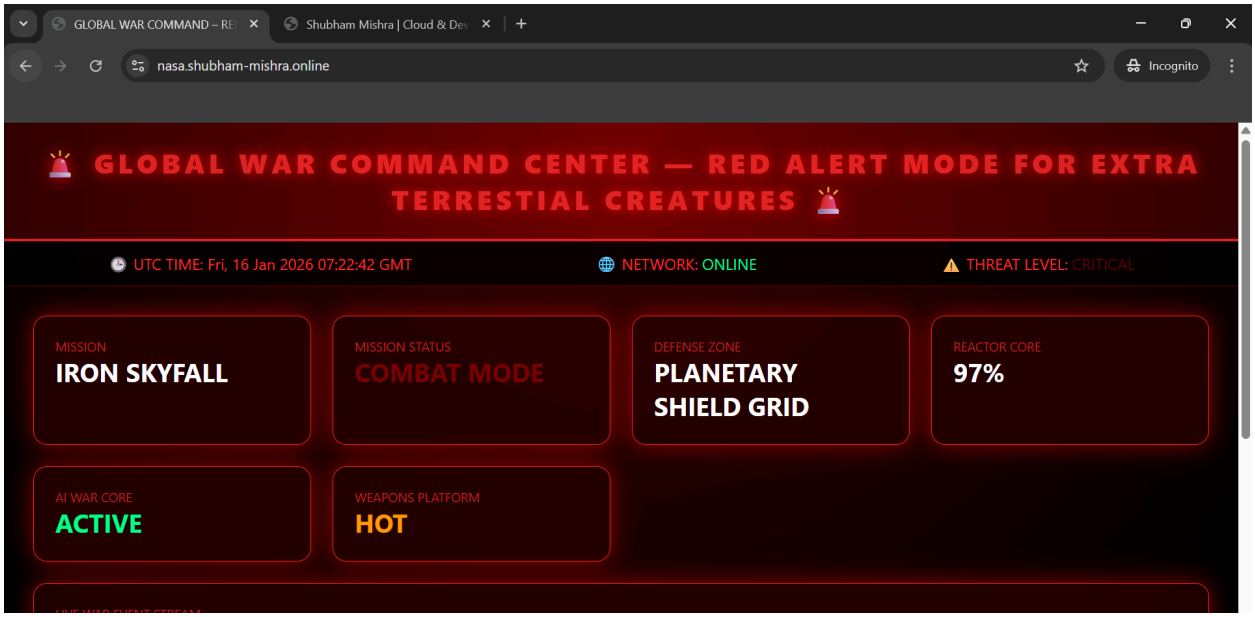


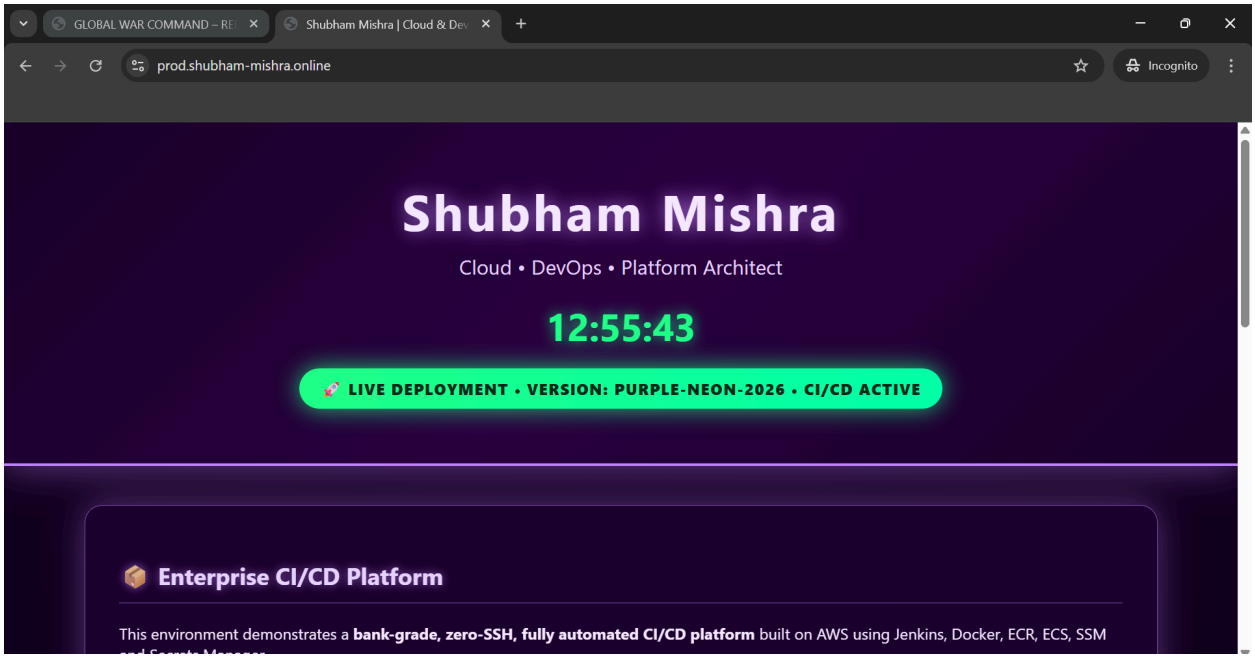
e) Now new task is getting created using updated image





g) We can se the new changes visible to end user using domain and traffic is also getting routed




GLOBAL WAR COMMAND – RE
Shubham Mishra | Cloud & De
prod.shubham-mishra.online
Incognito
Shubham Mishra
Cloud • DevOps • Platform Architect
12:55:43
LIVE DEPLOYMENT • VERSION: PURPLE-NEON-2026 • CI/CD ACTIVE
Enterprise CI/CD Platform
This environment demonstrates a bank-grade, zero-SSH, fully automated CI/CD platform built on AWS using Jenkins, Docker, ECR, ECS, SSM