Application Deployment

(Deploy the given React application to a production ready state)

Application:

Clone the below mentioned repository and deploy the application (Run application in port 3000).
Repo URL : https://github.com/Vennilavan12/Trend.git

Docker:

- Dockerize the application by creating Dockerfile
- Build an application and check output using docker image.

Terraform:

- Define infrastructure in main.tf to create VPC, IAM, EC2 with Jenkins, etc.
- Use terraform command to provision infrastructure.

DockerHub:

- Create a DockerHub repository.

Kubernetes:

- Setup Kubernetes in AWS EKS and Confirm EKS cluster is running.
- Write deployment and service YAML files.
- Deploy using kubectl via Jenkins.

Version Control:

- Push the codebase to a Git provider (GitHub).
- Add gitignore and dockerignore files and use CLI commands to push code.

Jenkins:

- Install Jenkins and necessary plugins (Docker, Git, Kubernetes, Pipeline) for build, push &deploy applications.
- Setup Github and jenkins integration using github webhook build trigger for auto build for every commit.
- Create a declarative pipeline script and pipeline project to build, push & deploy using CI-CD.

Monitoring:

Setup a monitoring system to check the health of the cluster or application (opensource) is highly appreciable.

Step 1: Local Project Setup and Version Control

1. Create a new folder for your project (e.g., ‘Project2'DevOps ')

2. Clone the original repository into this folder: git clone https://github.com/Vennilavan12/Trend.git

3. Navigate into the newly cloned project folder: cd Trend

4. First, check the current remote URL (it should show 'Vennilavan12/Trend.git'): git remote -v

5. Now, set the new remote URL to your repository : git remote set-url origin

   https://github.com/Saravanave/trend-app.git

6. Verify that the remote URL has been updated to your repository: git remote -v

7. Change the default branch name to 'main' for consistency (if it's not already): git branch -M main

8. Push the existing code from your local machine to your new GitHub repository: git push -u origin main

9. Add .gitignore and .dockerignore Files

10. Commit and Push the New Files

• .gitignore

Node modules

node_modules

.env

Build artifacts

build

dist

*.log

OS generated files

.DS_Store

.idea

.vscode

• .dockerignore

node_modules

.git

.gitignore

Dockerfile

Jenkinsfile

README.md

.env

• git commit -m "Add .gitignore and .dockerignore files"

• git push origin main

Step 2: Dockerize the Application

In this step, you will create a Docker image of your React application. This image will contain a web server (Nginx) that is configured to serve your application's static files from the dist folder. This makes your application portable and runnable in any environment that supports Docker.

2.1. Create the Dockerfile

```
FROM nginx:alpine
COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY dist /usr/share/nginx/html
EXPOSE 3000
CMD ["nginx", "-g", "daemon off;"]
```

2.2. Create the nginx.conf

```
server {
listen 3000;
server_name localhost;
location / {
root /usr/share/nginx/html;
index index.html index.htm;
try_files $uri $uri/ /index.html;
}
error_page 500 502 503 504 /50x.html;
location = /50x.html {
root /usr/share/nginx/html;
```

}

}

2.3. Build and Test the Docker Image Locally

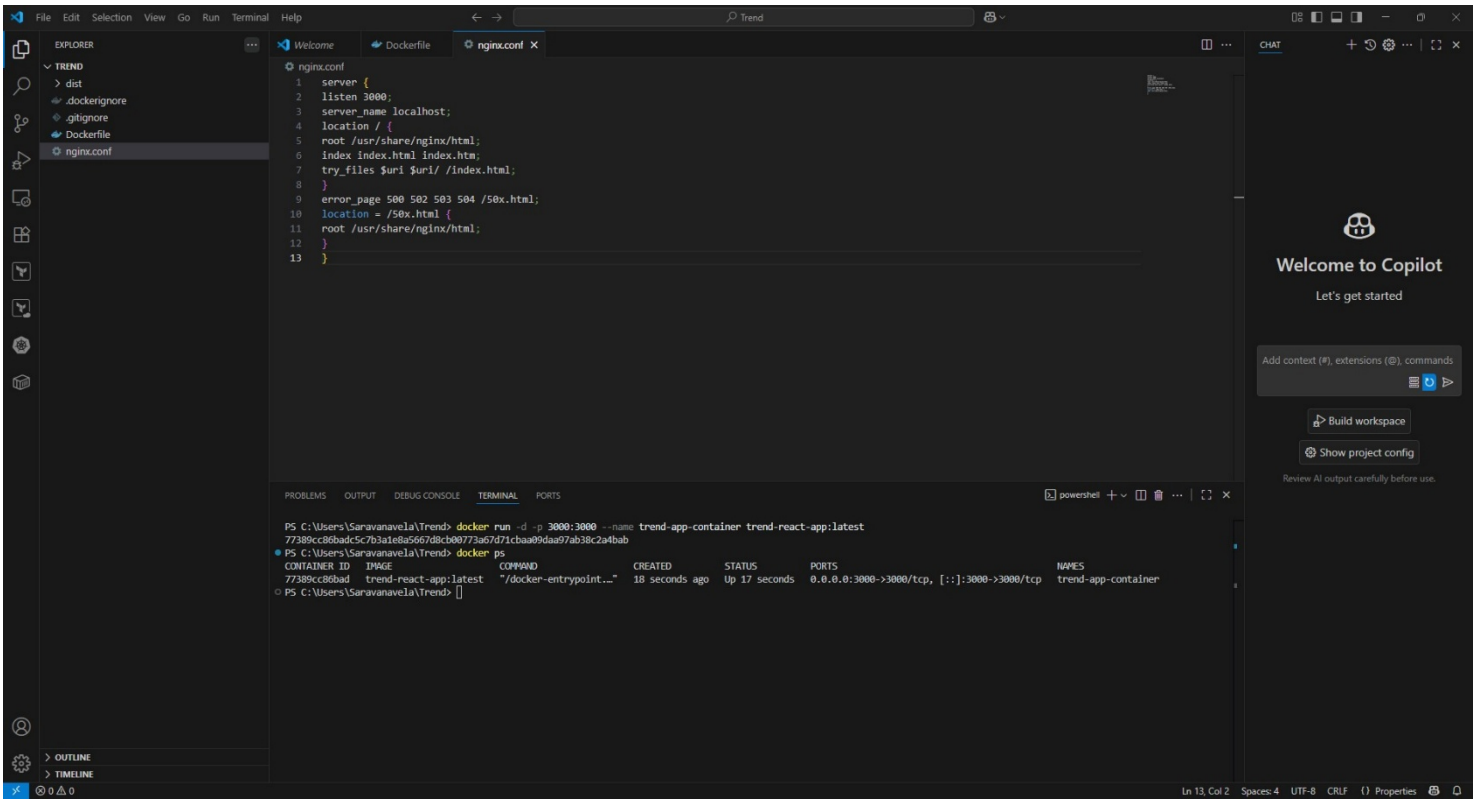
docker build -t trend-react-app:latest .

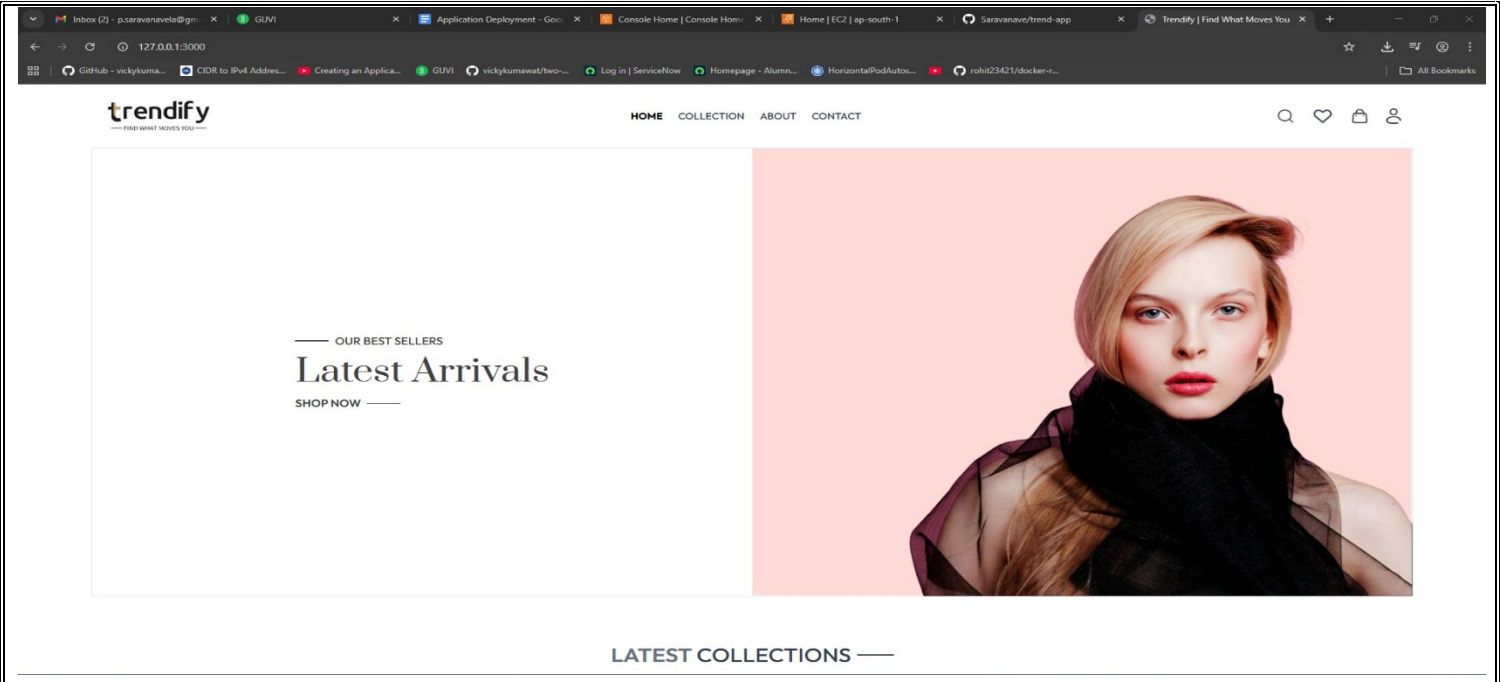
docker images

docker run -d -p 3000:3000 --name trend-app-container trend-react-app:latest

docker stop trend-app-container

docker rm trend-app-container



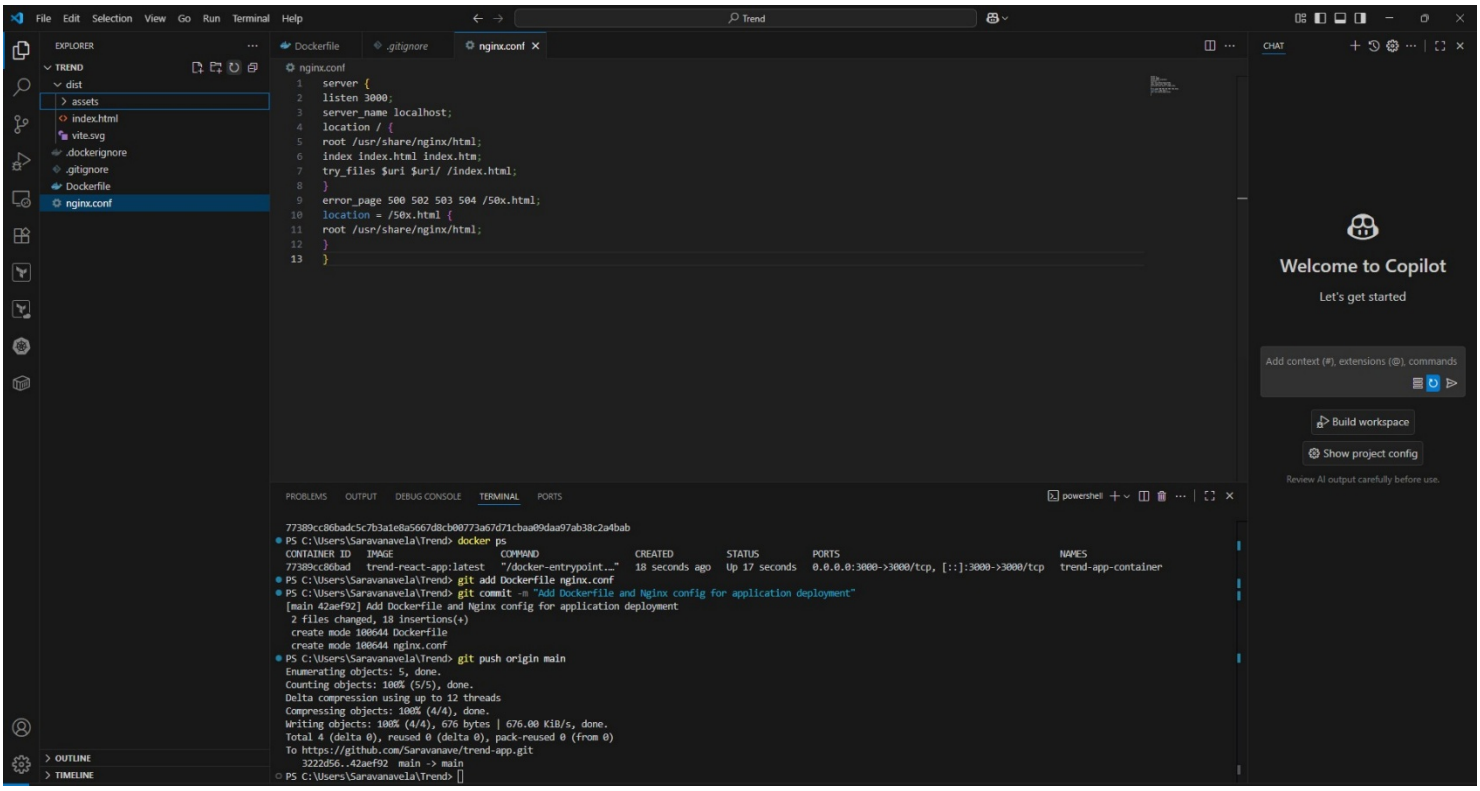


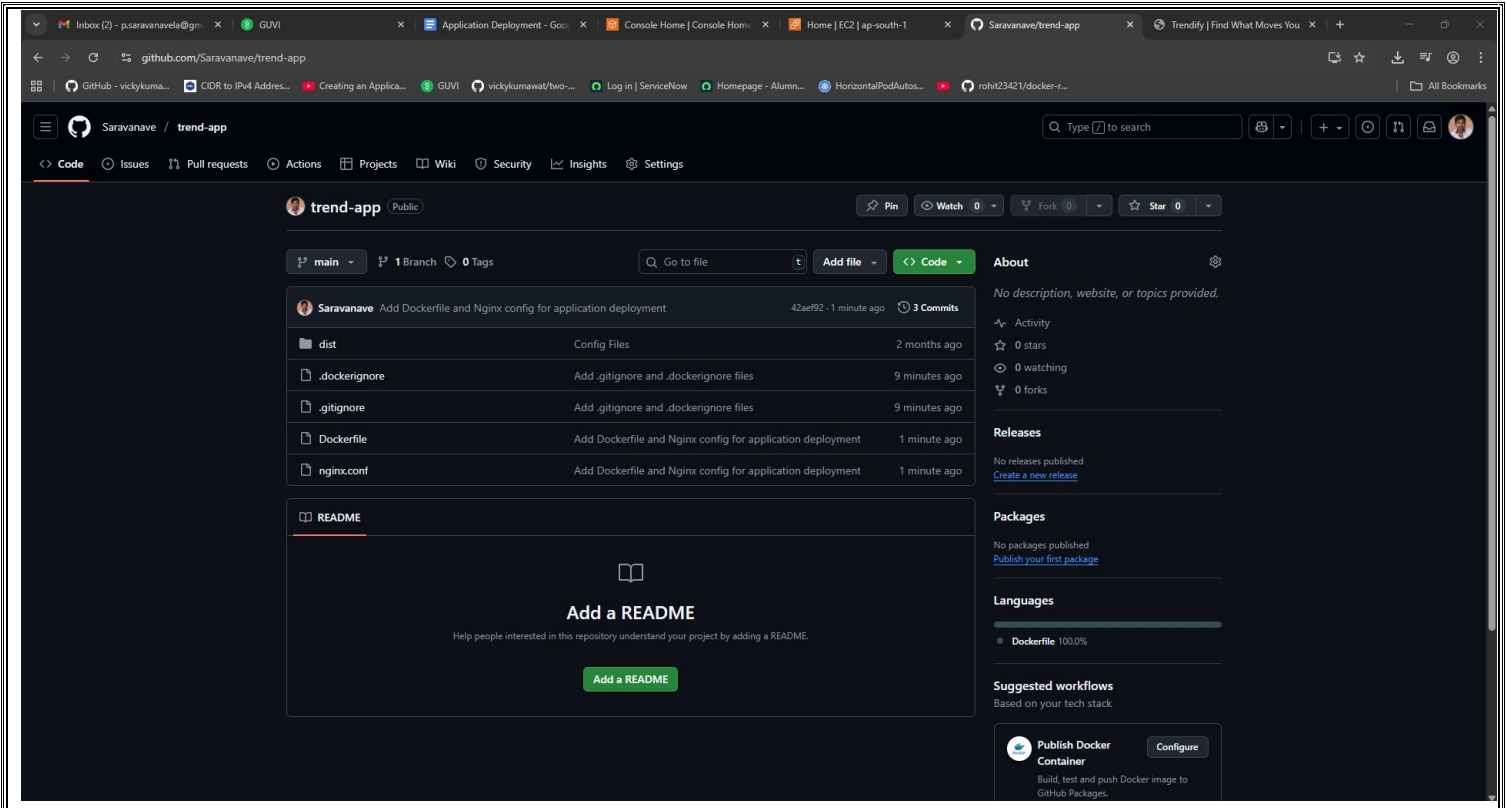
2.4. Commit and Push Docker Files

git add Dockerfile nginx.conf

git commit -m "Add Dockerfile and Nginx config for application deployment"

git push origin main





Step 3: Terraform Infrastructure

3.1. Create a Terraform Directory and main.tf

- # Go back to the parent directory (e.g., trend) : cd ..
- # Create a new directory for Terraform files
  mkdir terraform
  cd terraform
- #Inside this new terraform directory, create a file named: main.tf

3.2. Define Your Infrastructure in main.tf

```
terraform {

required_providers {

aws = {

source = "hashicorp/aws"

version = "6.10.0"

}

}
```

 # Remote state storage in S3

```
backend "s3" {
```

```
bucket = "your-unique-terraform-state-bucket" # REPLACE THIS NAME
key = "devops-project/terraform.tfstate"
region = "ap-south-1"
}
}
provider "aws" {
region = "ap-south-1"
}
# --- VPC and Subnet ---
resource "aws_vpc" "project_vpc" {
cidr_block = "10.0.0.0/16"
enable_dns_support = true
enable_dns_hostnames = true
tags = {
Name = "Project-VPC"
}
}
resource "aws_subnet" "public_subnet" {
vpc_id = aws_vpc.project_vpc.id
cidr_block = "10.0.1.0/24"
availability_zone = "ap-south-1a"
map_public_ip_on_launch = true
tags = {
Name = "Project-Public-Subnet"
}
}
resource "aws_subnet" "public_subnet_2" {
vpc_id = aws_vpc.project_vpc.id
cidr_block = "10.0.2.0/24" # New CIDR block for the second subnet
```

```
availability_zone = "ap-south-1b" # Different AZ
map_public_ip_on_launch = true
tags = {
Name = "Project-Public-Subnet-2"
}
}
resource "aws_internet_gateway" "project_igw" {
vpc_id = aws_vpc.project_vpc.id
tags = {
Name = "Project-IGW"
}
}
resource "aws_route_table" "project_rt" {
vpc_id = aws_vpc.project_vpc.id
route {
cidr_block = "0.0.0.0/0"
gateway_id = aws_internet_gateway.project_igw.id
}
tags = {
Name = "Project-RouteTable"
}
}
resource "aws_route_table_association" "project_rta" {
subnet_id = aws_subnet.public_subnet.id
route_table_id = aws_route_table.project_rt.id
}
# --- Security Group for Jenkins EC2 ---
resource "aws_security_group" "jenkins_sg" {
vpc_id = aws_vpc.project_vpc.id
```

```
name = "jenkins-ec2-sg"
description = "Security group for Jenkins EC2 instance"
ingress {
from_port = 22
to_port = 22
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
ingress {
from_port = 8080
to_port = 8080
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
ingress {
from_port = 50000
to_port = 50000
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
egress {
from_port = 0
to_port = 0
protocol = "-1"
cidr_blocks = ["0.0.0.0/0"]
}
tags = {
Name = "Jenkins-EC2-SG"
}
```

```
}
# --- IAM Role for Jenkins EC2 ---
resource "aws_iam_role" "ec2_instance_profile_role" {
name = "ec2-jenkins-role"
assume_role_policy = jsonencode({
Version = "2012-10-17"
Statement = [
{
Action = "sts:AssumeRole"
Effect = "Allow"
Principal = {
Service = "ec2.amazonaws.com"
}
},
]
})
tags = {
Name = "EC2-Jenkins-Role"
}
}
resource "aws_iam_role_policy_attachment" "ec2_policy_attachment" {
role = aws_iam_role.ec2_instance_profile_role.name
policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryFullAccess"
}
resource "aws_iam_role_policy_attachment" "eks_cluster_policy" {
role = aws_iam_role.ec2_instance_profile_role.name
policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
}
resource "aws_iam_role_policy_attachment" "eks_service_policy" {
```

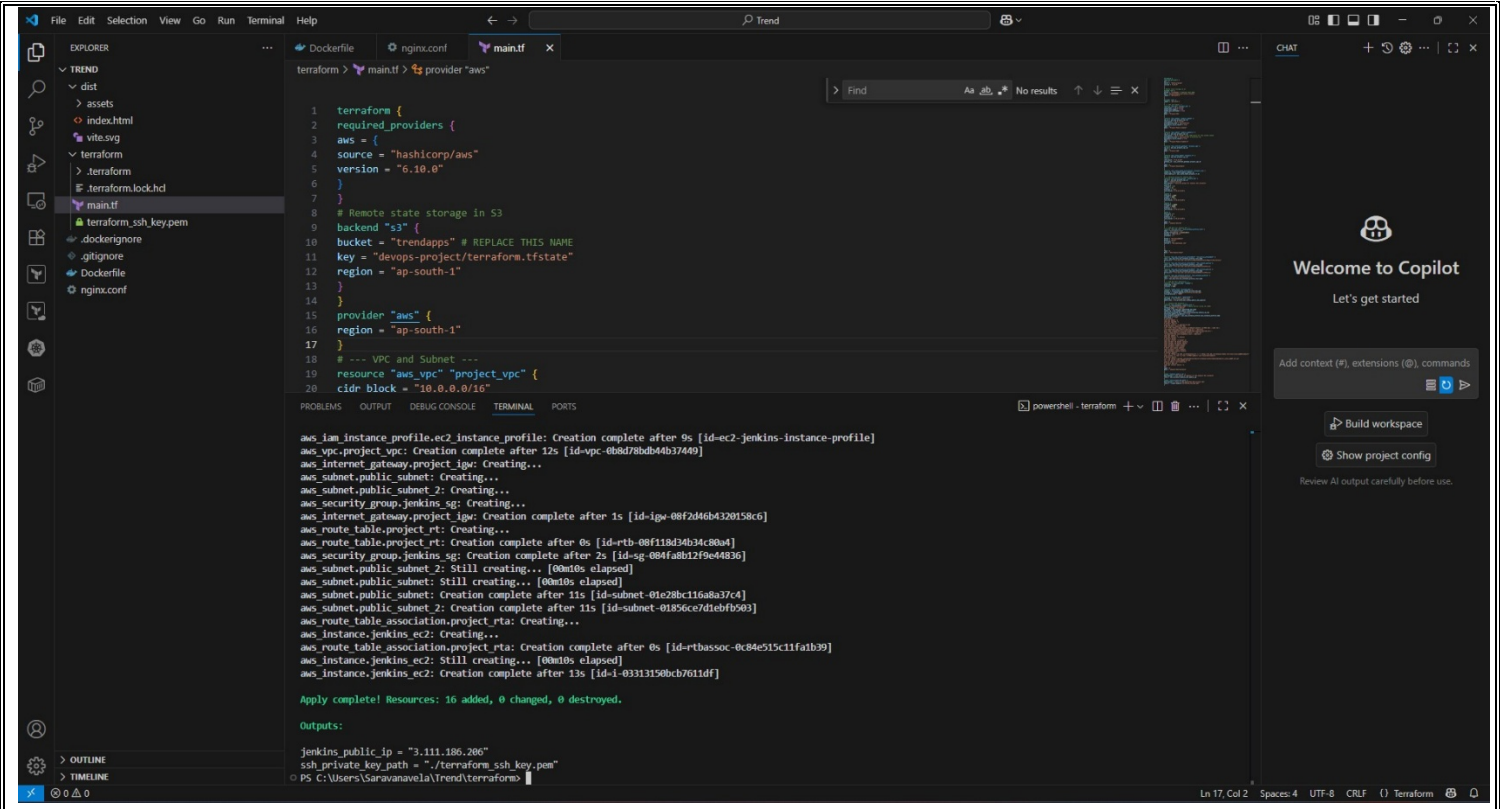
```
role = aws_iam_role.ec2_instance_profile_role.name
policy_arn = "arn:aws:iam::aws:policy/AmazonEKSServicePolicy"
}
resource "aws_iam_instance_profile" "ec2_instance_profile" {
name = "ec2-jenkins-instance-profile"
role = aws_iam_role.ec2_instance_profile_role.name
}
# --- SSH Key Pair Generation ---
resource "tls_private_key" "sshkey" {
algorithm = "RSA"
rsa_bits = 4096
}
resource "local_file" "private_key" {
content = tls_private_key.sshkey.private_key_pem
filename = "${path.module}/terraform_ssh_key.pem"
file_permission = "0400"
}
resource "aws_key_pair" "generated" {
key_name = "terraform-jenkins-key"
public_key = tls_private_key.sshkey.public_key_openssh
}
# --- Jenkins EC2 Instance ---
resource "aws_instance" "jenkins_ec2" {
ami = "ami-00bb6a80f01f03502" # Ubuntu Server 22.04 LTS (HVM)
instance_type = "t3.medium"
key_name = aws_key_pair.generated.key_name
subnet_id = aws_subnet.public_subnet.id
vpc_security_group_ids = [aws_security_group.jenkins_sg.id]
associate_public_ip_address = true
```

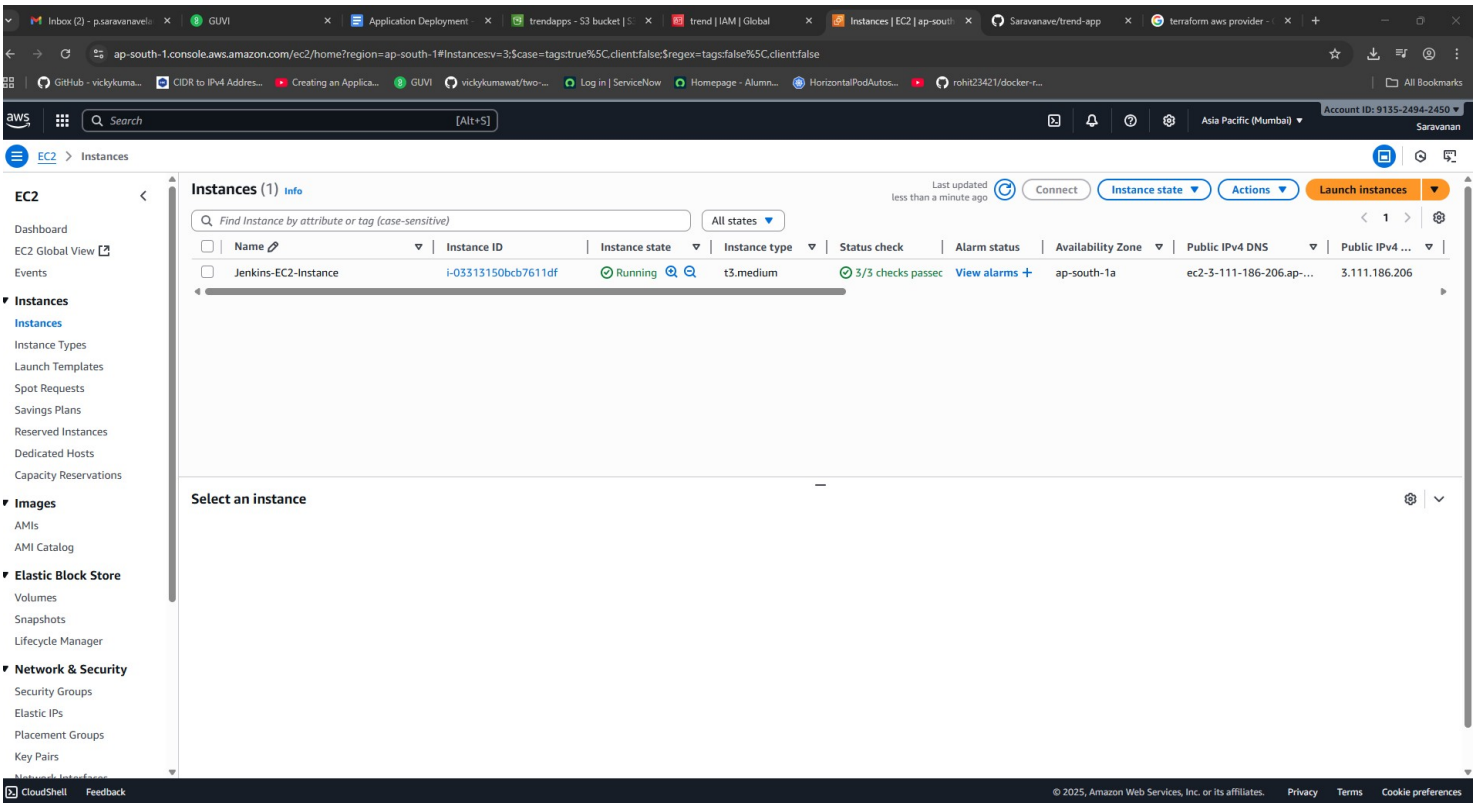
```
iam_instance_profile = aws_iam_instance_profile.ec2_instance_profile.name
user_data = <<-EOF
#!/bin/bash
sudo apt update -y
sudo apt upgrade -y
# Install Java 17
sudo apt install -y openjdk-17-jdk
# Add Jenkins repository key
curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian binary/" | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt update -y
sudo apt install -y jenkins
# Install Docker
sudo apt install -y docker.io
sudo usermod -aG docker jenkins
sudo usermod -aG docker ubuntu
sudo systemctl enable docker
sudo systemctl start docker
# Start and enable Jenkins
sudo systemctl start jenkins
sudo systemctl enable jenkins
# Install kubectl
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
# Install eksctl
```

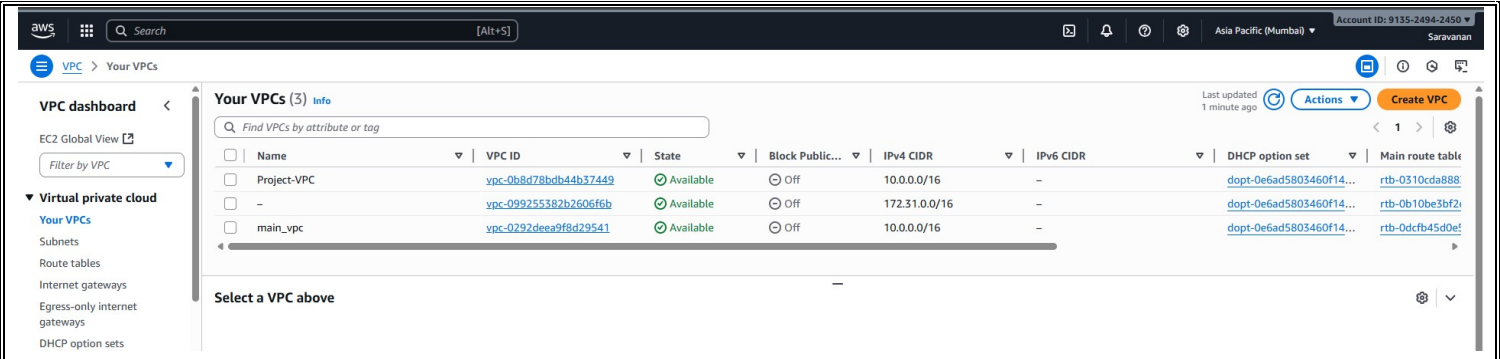
```
curl -LO
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_Linux_amd64.tar.gz"
tar -xzf eksctl_Linux_amd64.tar.gz
sudo mv eksctl /usr/local/bin
# Install AWS CLI
sudo apt install awscli -y
EOF
tags = {
Name = "Jenkins-EC2-Instance"
}
}
output "jenkins_public_ip" {
description = "The public IP address of the Jenkins EC2 instance"
value = aws_instance.jenkins_ec2.public_ip
}
output "ssh_private_key_path" {
description = "Path to the generated SSH private key"
value = "${path.module}/terraform_ssh_key.pem"
}
```

3.3. Run Terraform Commands

- terraform init
- terraform plan -out tfplan
- terraform apply "tfplan"


File Edit Selection View Go Run Terminal Help
Trend
EXPLORER
TREND
dist
assets
index.html
vite.svg
terraform
.terraform
.terraform.lock.hcl
main.tf
terraform_ssh_key.pem
.dockerignore
.gitignore
Dockerfile
nginx.conf
Dockerfile
nginx.conf
main.tf
terraform > main.tf > provider "aws"
terraform {
required_providers {
aws = {
source = "hashicorp/aws"
version = "6.10.0"
}
}
# Remote state storage in S3
backend "s3" {
bucket = "trendapps" # REPLACE THIS NAME
key = "devops-project/terraform.tfstate"
region = "ap-south-1"
}
}
provider "aws" {
region = "ap-south-1"
}
# --- VPC and Subnet ---
resource "aws_vpc" "project_vpc" {
cidr_block = "10.0.0.0/16"
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell - terraform
aws_iam_instance_profile.ec2_instance_profile: Creation complete after 9s [id=ec2-jenkins-instance-profile]
aws_vpc.project_vpc: Creation complete after 12s [id=vpc-0b8d78bdb44b37449]
aws_internet_gateway.project_igw: Creating...
aws_subnet.public_subnet: Creating...
aws_subnet.public_subnet_2: Creating...
aws_security_group.jenkins_sg: Creating...
aws_internet_gateway.project_igw: Creation complete after 1s [id=igw-08f2d46b4320158c6]
aws_route_table.project_rt: Creating...
aws_route_table.project_rt: Creation complete after 0s [id=rtb-08f118d34b34c80a4]
aws_security_group.jenkins_sg: Creation complete after 2s [id=sg-084fa8b12f9e44836]
aws_subnet.public_subnet_2: Still creating... [00m10s elapsed]
aws_subnet.public_subnet: Still creating... [00m10s elapsed]
aws_subnet.public_subnet: Creation complete after 11s [id=subnet-01e28bc116a8a37c4]
aws_subnet.public_subnet_2: Creation complete after 11s [id=subnet-01856ce7d1ebfb503]
aws_route_table_association.project_rta: Creating...
aws_instance.jenkins_ec2: Creating...
aws_route_table_association.project_rta: Creation complete after 0s [id=rtbassoc-0c84e515c11fa1b39]
aws_instance.jenkins_ec2: Still creating... [00m10s elapsed]
aws_instance.jenkins_ec2: Creation complete after 13s [id=i-03313150bcb7611df]
Apply complete! Resources: 16 added, 0 changed, 0 destroyed.
Outputs:
jenkins_public_ip = "3.111.186.206"
ssh_private_key_path = "./terraform_ssh_key.pem"
PS C:\Users\Saravanavela\Trend\terraform>
CHAT
Welcome to Copilot
Let's get started
Add context (#), extensions (@), commands
Build workspace
Show project config
Review AI output carefully before use.
OUTLINE
TIMELINE
Ln 17, Col 2 Spaces: 4 UTF-8 CRLF Terraform


Inbox (2) - p.saravanavela
GUVI
Application Deployment
trendapps - S3 bucket | S
trend | IAM | Global
Instances | EC2 | ap-south
Saravanave/trend-app
terraform aws provider -
ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#Instances:v=3;$case=tags:true%5C,client:false;$regex=tags:false%5C,client:false
GitHub - vickykuma...
CIDR to IPv4 Addres...
Creating an Applica...
GUVI
vickykumawat/two-...
Log in | ServiceNow
Homepage - Alumn...
HorizontalPodAutos...
rohit23421/docker-r...
All Bookmarks
aws
Search
[Alt+S]
Asia Pacific (Mumbai)
Account ID: 9135-2494-2450
Saravanan
EC2 > Instances
EC2
Dashboard
EC2 Global View
Events
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Network & Security
Security Groups
Elastic IPs
Placement Groups
Key Pairs
Instances (1) Info
Last updated less than a minute ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Public IPv4 ...
Jenkins-EC2-Instance
i-03313150bcb7611df
Running
t3.medium
3/3 checks passec
View alarms
ap-south-1a
ec2-3-111-186-206.ap-...
3.111.186.206
Select an instance
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



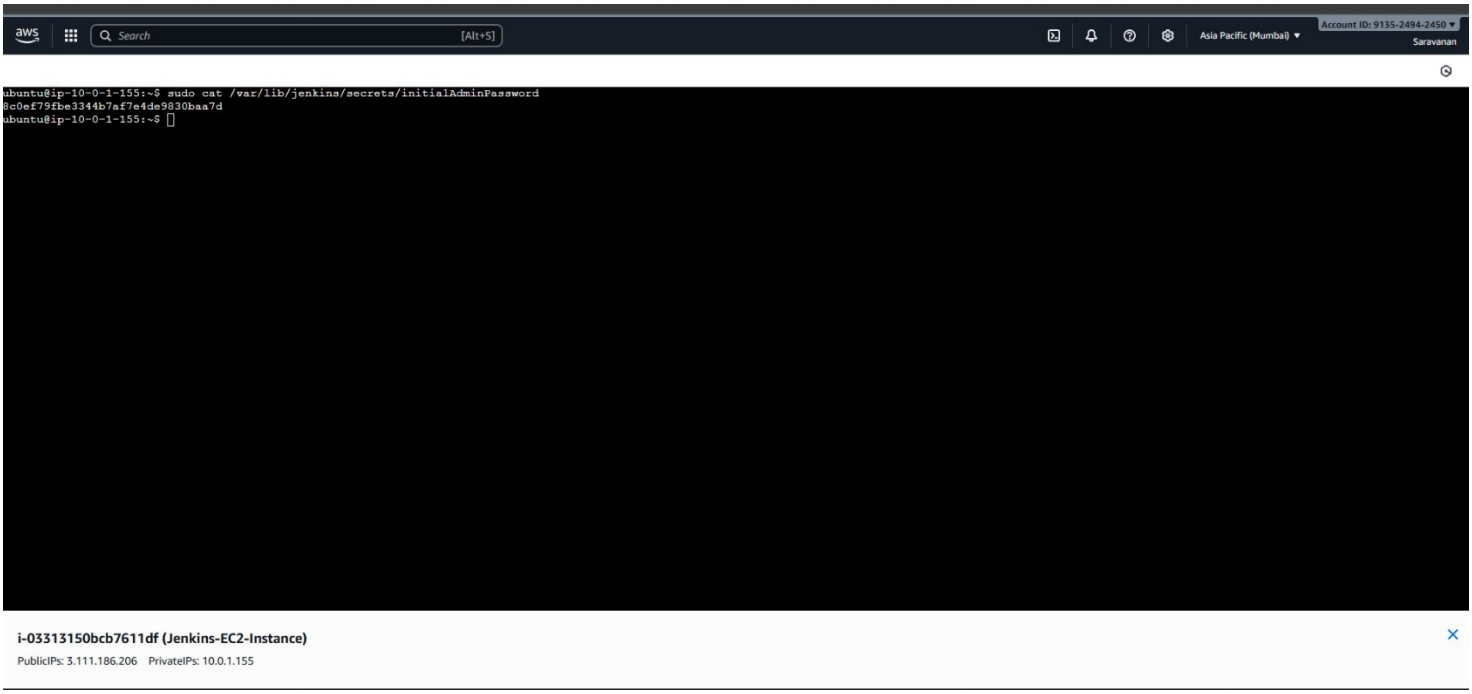
Step 4: Jenkins Setup and Configuration (Jenkins server configuration and integration with GitHub and DockerHub)

4.1. Access Jenkins and Complete Initial Setup

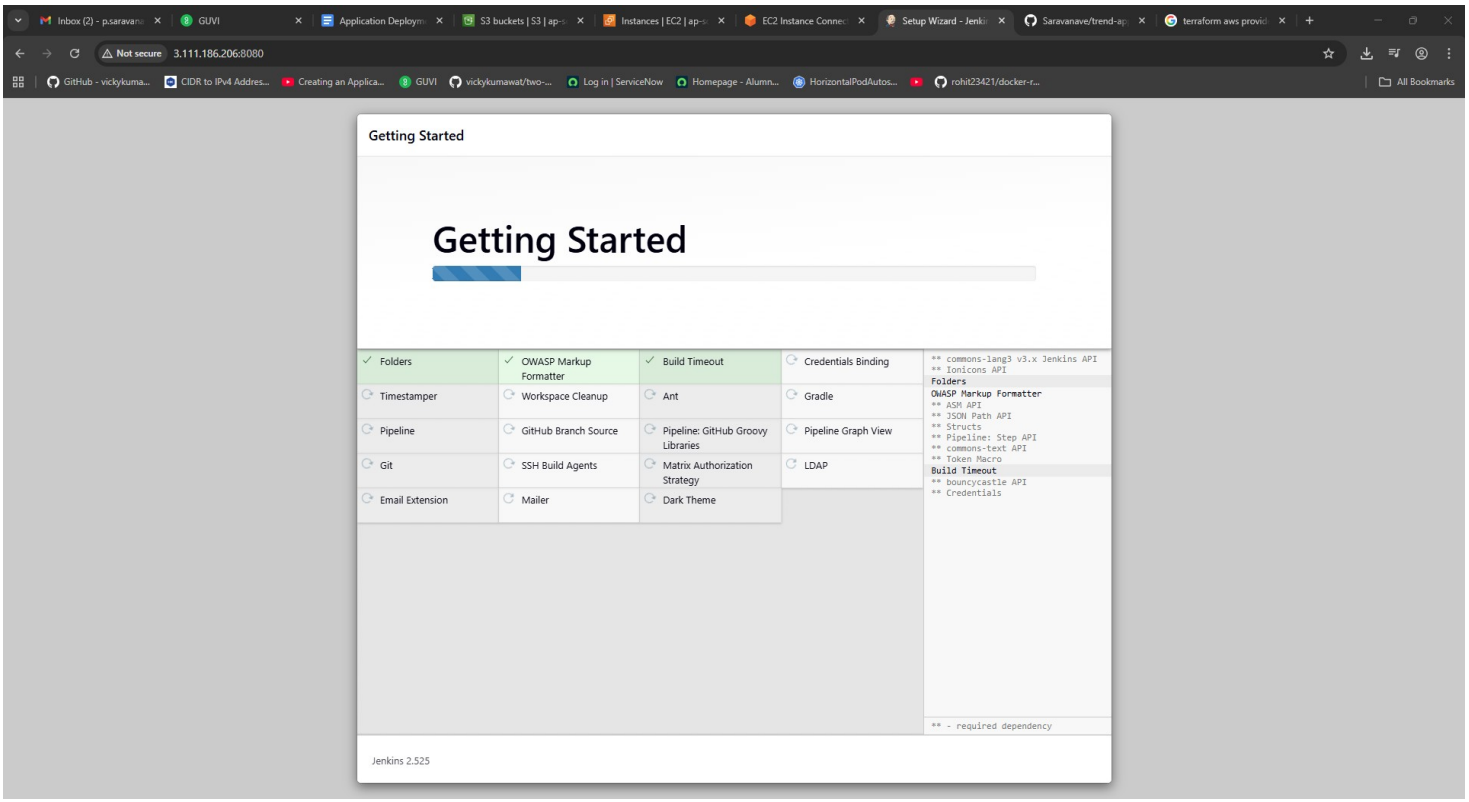
Find Your Jenkins Public IP:

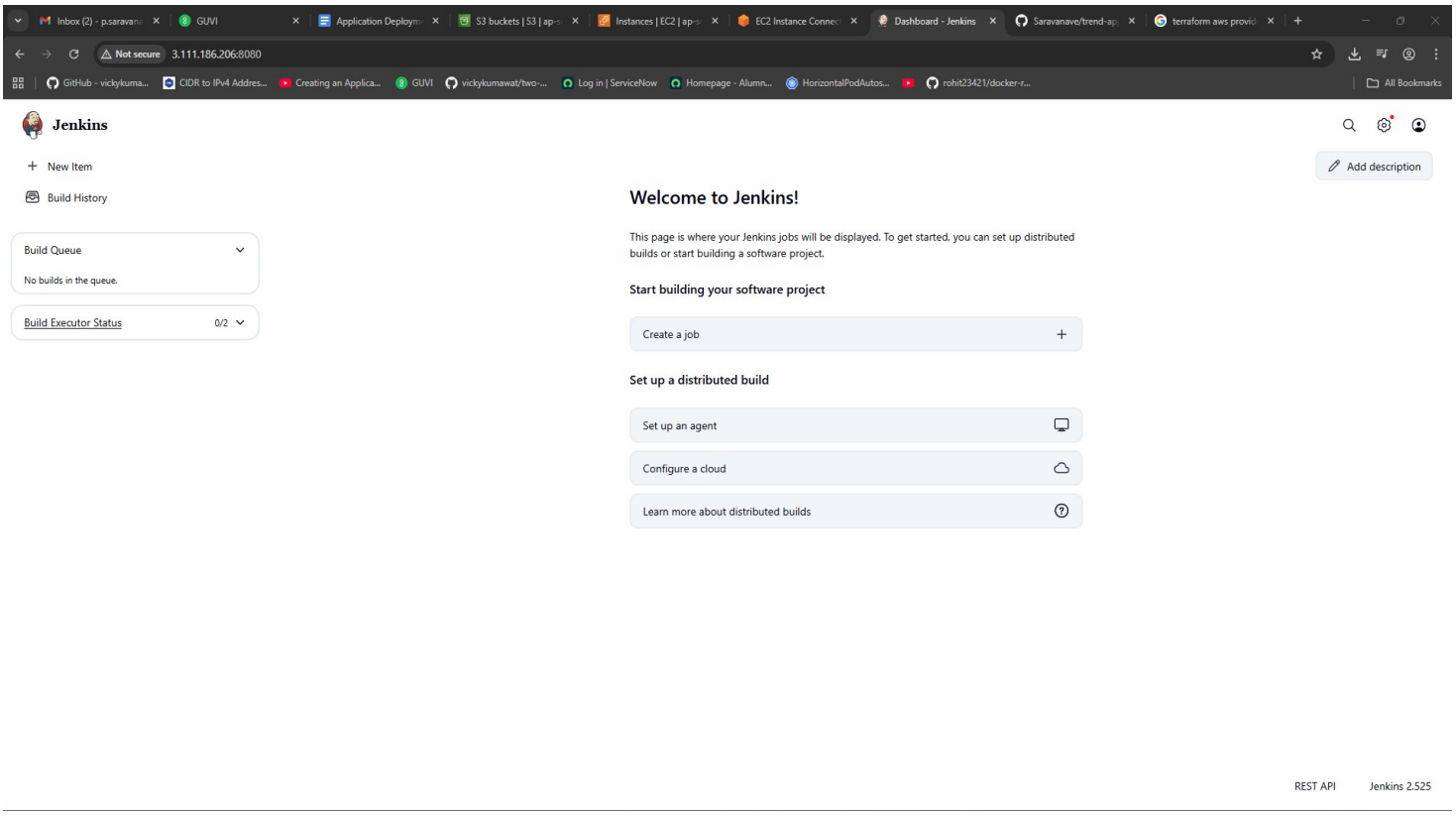
Access the Jenkins UI:

Retrieve the Initial Admin Password: sudo cat /var/lib/jenkins/secrets/initialAdminPassword



Complete the Jenkins Wizard:


Not secure 3.111.186.206:8080
Getting Started
Getting Started
Folders
OWASP Markup Formatter
Build Timeout
Credentials Binding
Timestamper
Workspace Cleanup
Ant
Gradle
Pipeline
GitHub Branch Source
Pipeline: GitHub Groovy Libraries
Pipeline Graph View
Git
SSH Build Agents
Matrix Authorization Strategy
LDAP
Email Extension
Mailer
Dark Theme
** commons-lang3 v3.x Jenkins API
** Ionicons API
Folders
OWASP Markup Formatter
** ASM API
** JSON Path API
** Structs
** Pipeline: Step API
** commons-text API
** Token Macro
Build Timeout
** bouncycastle API
** Credentials
** - required dependency
Jenkins 2.525


Not secure 3.111.186.206:8080
Jenkins
New Item
Build History
Build Queue
No builds in the queue.
Build Executor Status 0/2
Add description
Welcome to Jenkins!
This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.
Start building your software project
Create a job
Set up a distributed build
Set up an agent
Configure a cloud
Learn more about distributed builds
REST API
Jenkins 2.525

4.2. Install Necessary Plugins: go to Manage Jenkins -> Manage Plugins.-> Click on the Available tab.

- ✓ GitHub Plugin
- ✓ GitHub Branch Source Plugin
- ✓ Docker Pipeline
- ✓ Kubernetes CLI

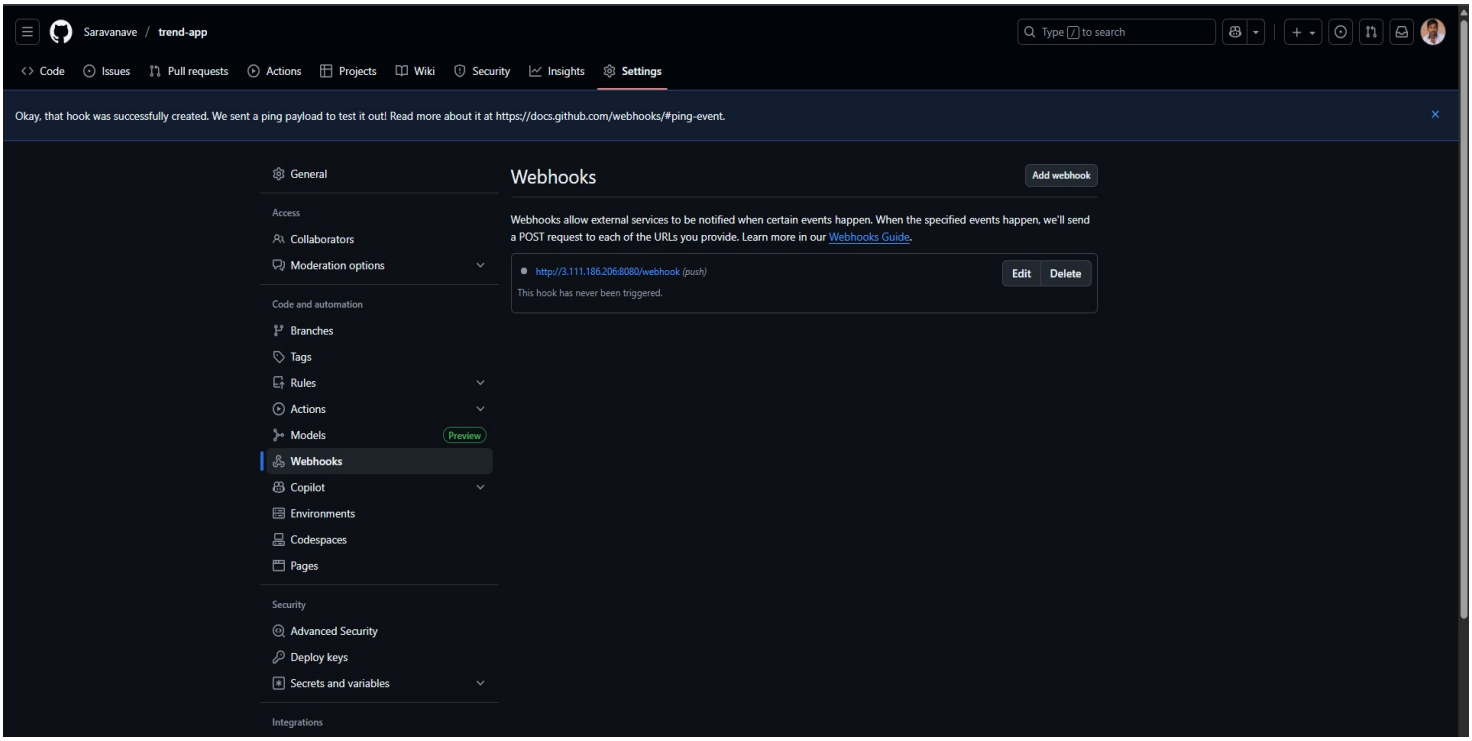
4.3. Configure GitHub Webhook Trigger

4.3.1 In Jenkins:

- ❖ Go to Manage Jenkins -> Configure System
- ❖ Scroll down to the GitHub section
- ❖ Click Add GitHub Server and select GitHub Server
- ❖ Click Manage Hooks and ensure the box Manage hooks for this server is checked
- ❖ Click Save.

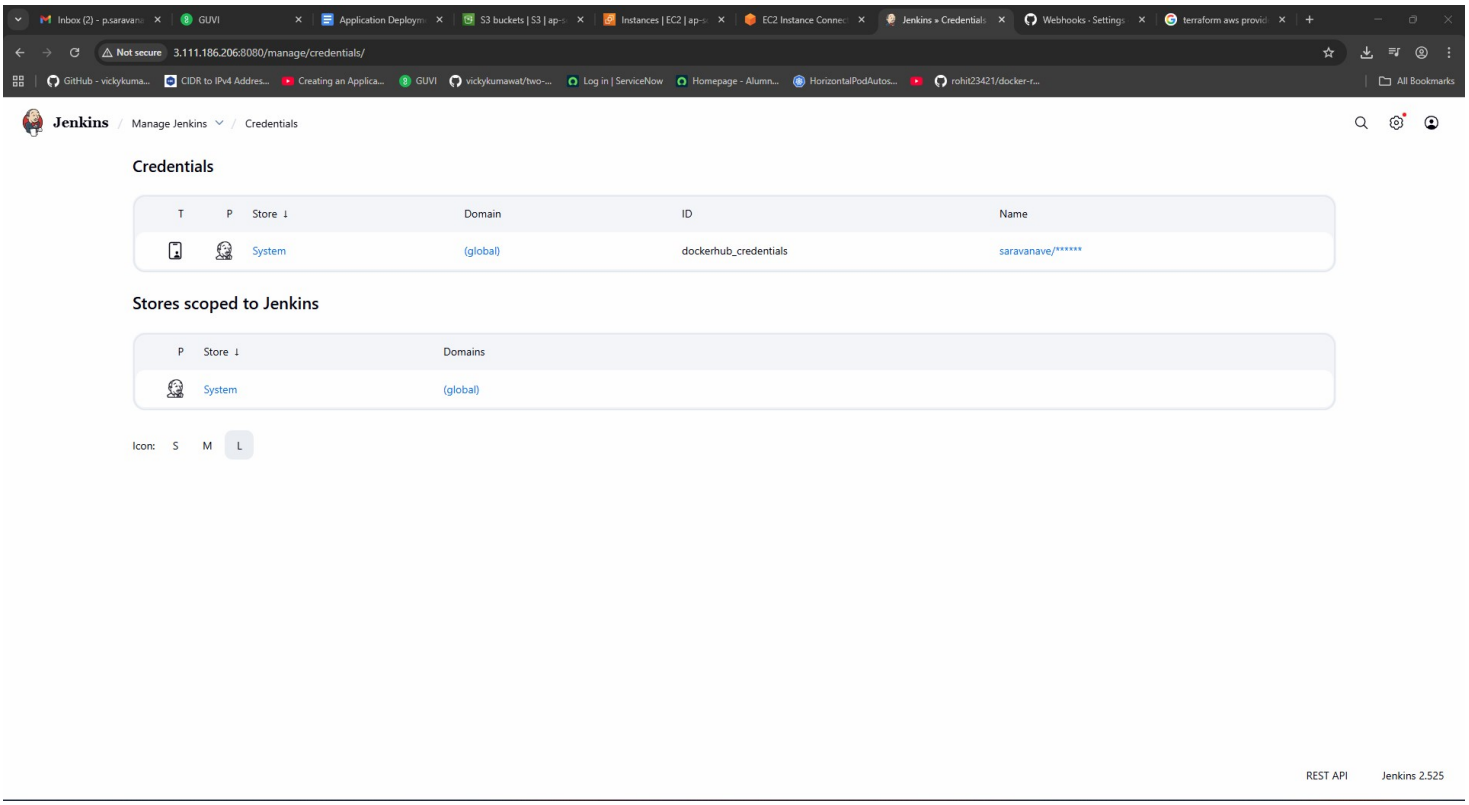
4.3.2 In GitHub:

- ❖ Go to your repository: https://github.com/Saravanave/trend-app.git
- ❖ Click on Settings -> Webhooks->Click Add webhook
- ❖ **Payload URL:** Enter http://<YOUR_JENKINS_PUBLIC_IP>:8080/github-webhook/
- ❖ **Content type:** Select application/json.
- ❖ **Which events would you like to trigger this webhook?:** Select Just the push event.
- ❖ Make sure Active is checked.
- ❖ Click Add webhook.



4.4. Store DockerHub Credentials in Jenkins:

- In Jenkins, go to Manage Jenkins -> Manage Credentials.
- Click on (global) -> Add Credentials.
- From the dropdown, select Username with password.
- Username: Your DockerHub username.
- Password: Your DockerHub password.
- ID: dockerhub_credentials (This ID is very important; you will use it in your Jenkinsfile later).
- Description: DockerHub Credentials
- Click Create.



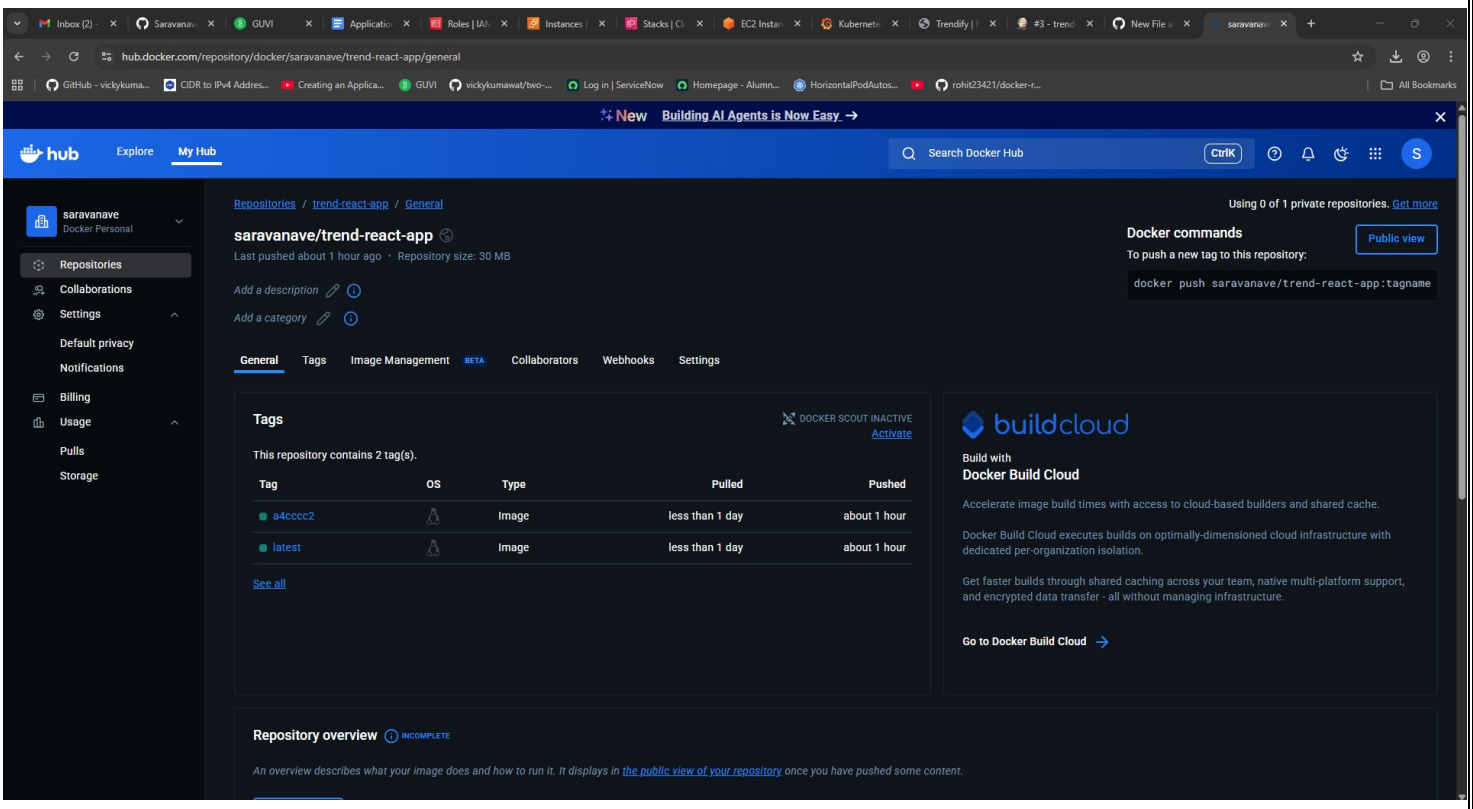
Step 5: DockerHub Repository:

This is a quick but crucial step. Your Jenkins pipeline will build a Docker image of your React application and then push that image to a repository. You need to create this repository on DockerHub first.

1. **Navigate to DockerHub:**
    a. Open your web browser and go to https://hub.docker.com/
    b. Log in to your DockerHub account if you haven't already.
2. **Create a New Repository:**
    a. In the top navigation bar, click on **Repositories**
    b. Click the **Create Repository** button
3. **Fill in the Repository Details:**
    - **Name:** This name needs to be consistent with what you will use in your Jenkins pipeline. A good naming convention is your_dockerhub_username/application_name.

- Use the name: your_dockerhub_username/trend-react-app
- Replace your_dockerhub_username with your actual DockerHub username.
  - **Description:** You can add a short description, like Docker image for the React Trend application.
  - **Visibility:** Set this to Public. This is important because it allows the Kubernetes cluster to pull the image without needing extra authentication credentials, simplifying the deployment process.

4. Click the **Create** button



Step 6: Kubernetes Setup (AWS EKS)

Your Terraform user_data script has already installed awscli, eksctl, and kubectl for you. However, you need to ensure the IAM role of the Jenkins EC2 instance has the right permissions to create and manage EKS resources.

1. Verify IAM Role Permissions:

- Go to your AWS IAM console.
- Go to Roles and find the role named ec2-jenkins-role.
- Click on the role and check the Permissions tab.
- Ensure the following policies are attached:

a. AmazonEKSClusterPolicy
b. AmazonEKSServicePolicy
c. AmazonEC2ContainerRegistryFullAccess (Even though we're using DockerHub, this is a standard permission for EKS node groups).

Create policy ->jsontab

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:CreateCluster",
        "eks:DescribeCluster",
        "eks:UpdateClusterConfig",
        "eks:DeleteCluster",
        "eks:TagResource",
        "eks:UntagResource",
        "eks:ListTagsForResource",
        "eks:DescribeUpdate",
        "eks:ListUpdates",
        "eks:UpdateClusterVersion",
        "eks:CreateNodegroup",
        "eks:UpdateNodegroupConfig",
        "eks:DeleteNodegroup",
        "eks:DescribeNodegroup",
        "eks:ListNodegroups",
        "eks:DescribeClusterVersions",
        "iam:GetRole",
        "iam:ListRoles",
        "iam:CreateRole",
        "iam:AttachRolePolicy",
        "iam:DeleteRole",
        "iam:DetachRolePolicy",
        "iam:PassRole",
        "ec2:DescribeKeyPairs",
        "ec2:CreateTags",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:CreateSecurityGroup",
        "ec2:RevokeSecurityGroupEgress",
        "ec2:AuthorizeSecurityGroupEgress",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:DeleteSecurityGroup",
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:AttachNetworkInterface",
```

```
        "ec2:DetachNetworkInterface",
        "ec2:RunInstances",
        "ec2:DescribeInstances",
        "ec2:TerminateInstances",
        "ec2:DescribeImages",
        "ec2:DescribeInstanceStatus",
        "ec2:DescribeVolumes",
        "ec2:CreateVolume",
        "ec2:AttachVolume",
        "ec2:DetachVolume",
        "ec2:DeleteVolume",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ssm:GetParameter"
      ],
      "Resource": "*"
    }
  ]
}
```

- Add permissions -> Attach policies and add them.

6.2. Create EKS Cluster Configuration File (cluster.yaml)

This file will describe the EKS cluster you want to create. It's best practice to define it in a file rather than passing all parameters in the command line.

1. While you are logged into the Jenkins EC2 instance, create a new file named cluster.yaml in your
   home directory: nano cluster.yaml
2. Copy and paste the following content into the file. **You must replace the placeholders for the VPC and subnet IDs with the ones created by Terraform in Step 3.**

To find these IDs, go to your AWS Console:

• VPC ID: Go to VPC -> Your VPCs and find the one named Project-VPC. Copy its ID.

• Subnet ID: Go to VPC -> Subnets and find the one named Project-Public-Subnet. Copy its ID.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata: null
name: my-trend-cluster
region: ap-south-1
version: '1.29'
vpc: null
id: <YOUR_VPC_ID>
subnets: null
public: null
ap-south-1a:
  id: <YOUR_PUBLIC_SUBNET_ID>
ap-south-1b:
  id: <YOUR_PUBLIC_SUBNET_2_ID>
managedNodeGroups:
  - name: my-trend-nodes
instanceType: t3.medium
desiredCapacity: 2
minSize: 1
maxSize: 3
volumeSize: 20
ssh: null
allow: true
publicKeyPath: /home/ubuntu/.ssh/id_rsa.pub
```

6.4. Create the EKS Cluster

Now, use eksctl with the configuration file to create the cluster.

Run the following command:

eksctl create cluster -f cluster.yaml+

6.5. Verify Cluster Creation and Configure kubectl

After eksctl finishes, you need to configure kubectl to communicate with the new cluster.

Before that configure AWS CLI in your instance

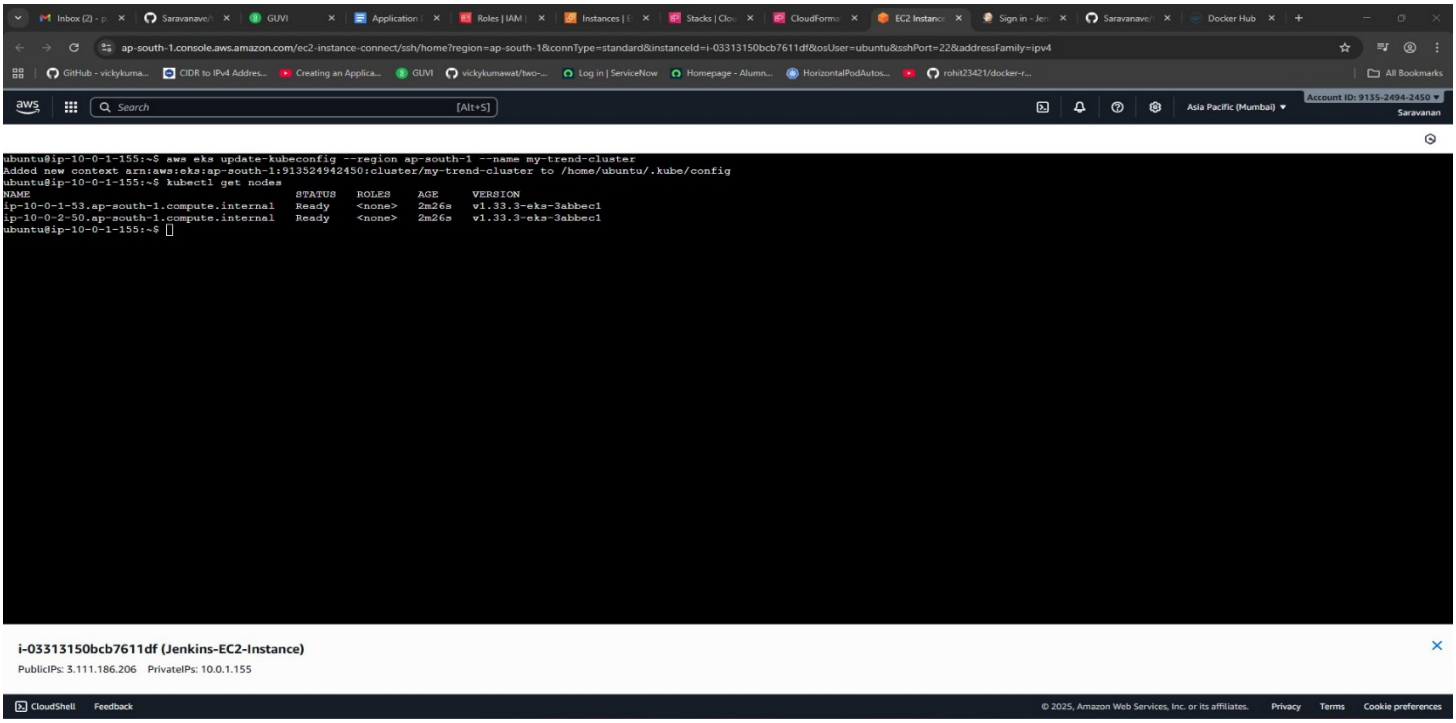
- sudo apt update && sudo apt upgrade -y
- sudo apt install unzip curl -y
- curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
- unzip awscliv2.zip
- sudo ./aws/install
- aws –version

1. Run the following command to update your kubeconfig file with the cluster information:

aws eks update-kubeconfig --region ap-south-1 --name my-trend-cluster

2. Now, verify that kubectl can see the cluster nodes:

kubectl get nodes



Step 7: Kubernetes Deployment Manifests

Now that you have a running Kubernetes cluster, you need to tell it how to deploy your application. You will do this by creating two YAML files: a Deployment to manage your application's pods and a Service to expose your application to the internet.

You will create these files on the Jenkins EC2 instance because your Jenkins pipeline will need to use them

7.1. Create the deployment.yaml File*: nano deployment.yaml*

```
apiVersion: apps/v1

kind: Deployment

metadata: null

name: trend-app-deployment

labels: null

app: trend-app

spec: null

replicas: 2

selector: null

matchLabels:

  app: trend-app

template:

  metadata: null

  labels:

    app: trend-app

  spec: null

  containers:

    - name: trend-app-container

image: 'saravanave/trend-react-app:latest'
```

ports:

- containerPort: 3000

7.2. Create the service.yaml

This file creates a Service that provides a stable IP address and DNS name for your application. We will use the LoadBalancer type, which automatically provisions an AWS Network Load Balancer (NLB) to expose your application to the public internet.

apiVersion: v1

kind: Service

metadata:

name: trend-app-service

labels:

app: trend-app

annotations:

service.beta.kubernetes.io/aws-load-balancer-type: "nlb"

spec:

selector:

app: trend-app

type: LoadBalancer

ports:

- protocol: TCP

port: 80

targetPort: 3000

it's great that your AWS EKS cluster is now successfully created and operational

7.3. Initial Deployment and Verification

It's a good practice to test these files manually before putting them in your Jenkins pipeline.

1. Deploy your application and service:

kubectl apply -f deployment.yaml

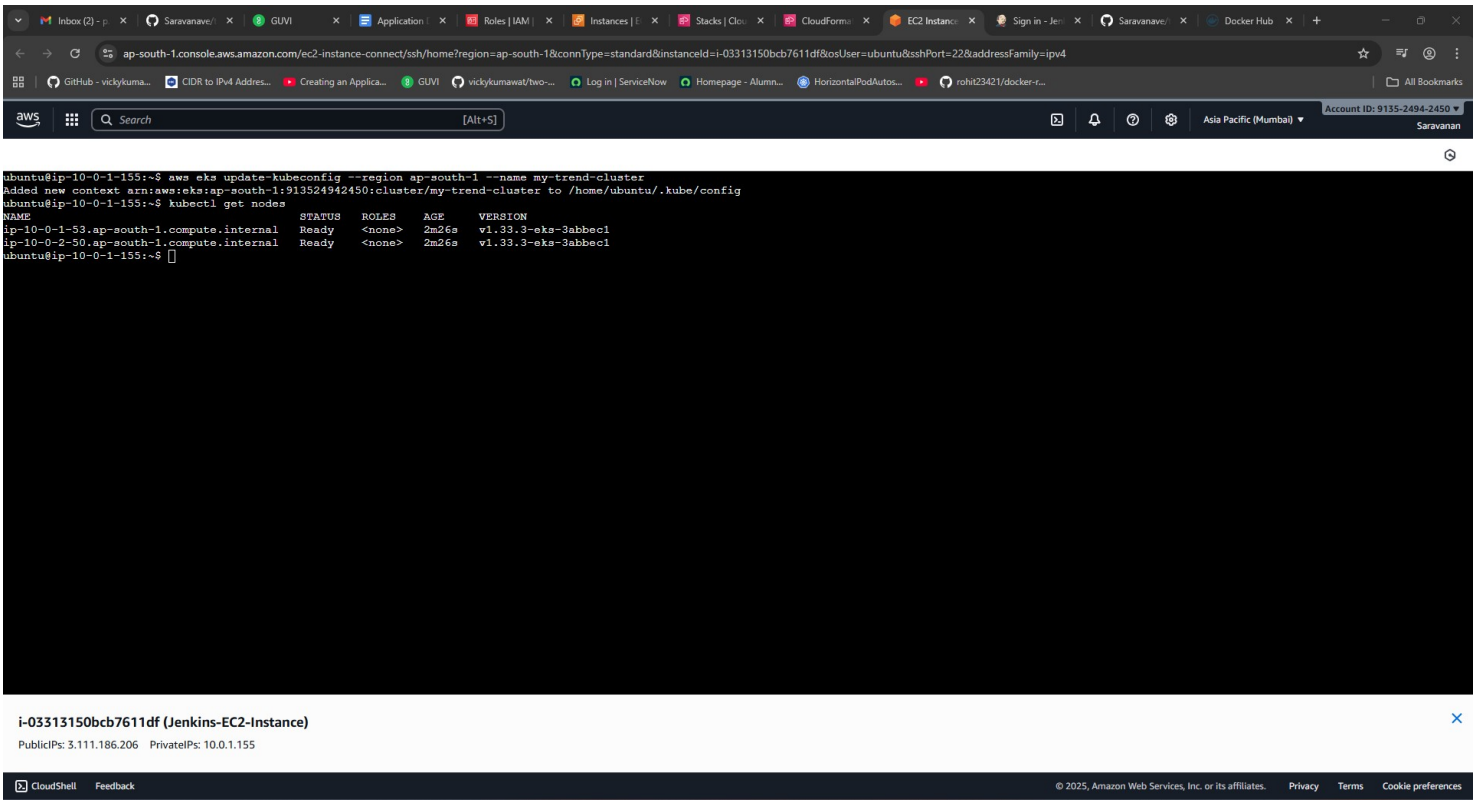
kubectl apply -f service.yaml

2. Verify the deployment:

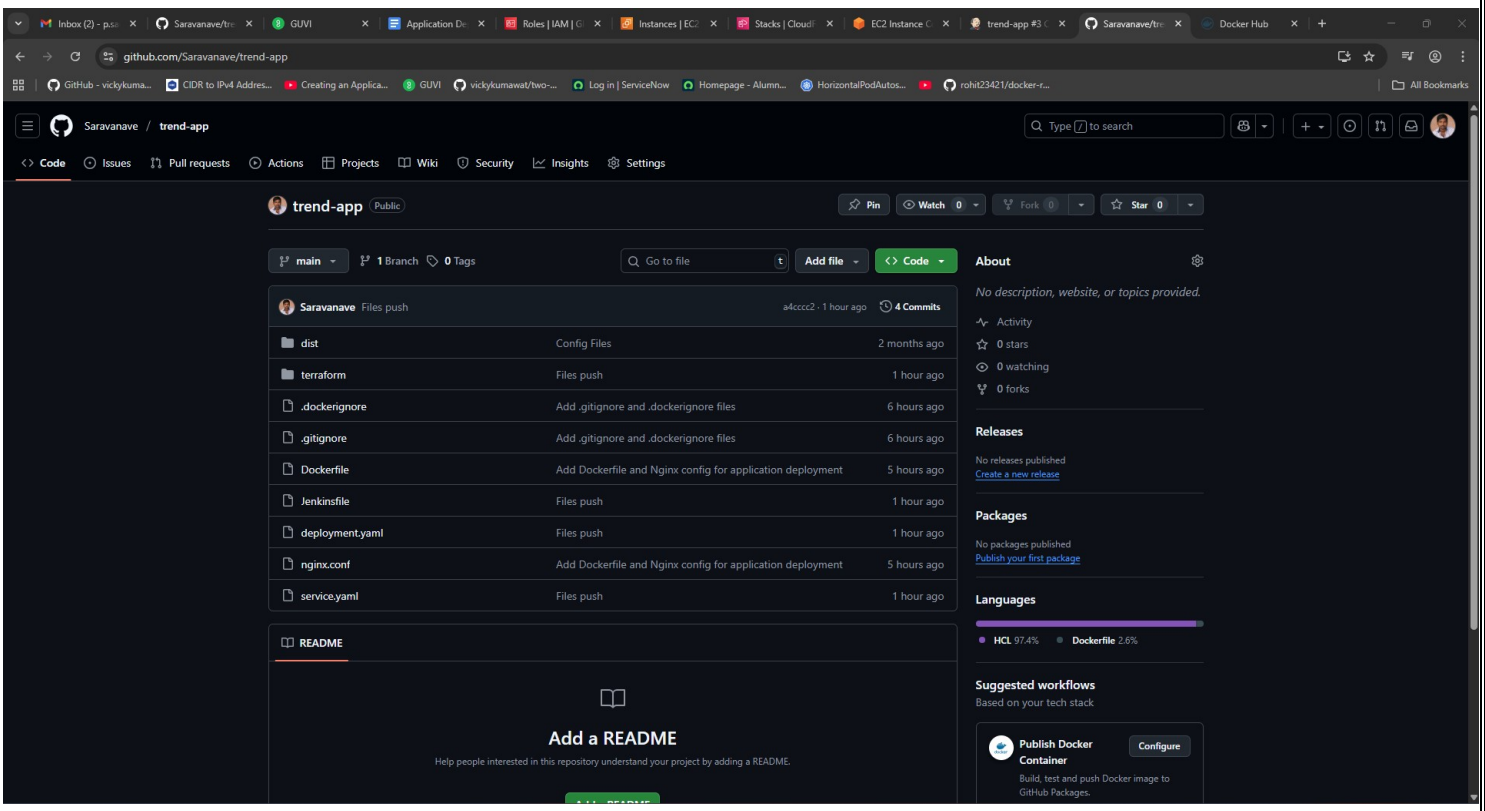
kubectl get deployments

kubectl get pods

kubectl get services



7.4. Push to GitHub



Step 8: Jenkins CI/CD Pipeline: The **Jenkinsfile** is a text file that defines the steps of your pipeline. It lives in the root of your application's GitHub repository.

1. **On your local machine**, navigate to the root of your Trend project directory. This is the directory
   where you cloned the code and created the Docker files.
2. Create the **Jenkinsfile**
3. Paste the following content. This script has stages for building the Docker image, pushing it to DockerHub, and deploying it to Kubernetes. Make sure to **replace Saravanave/trend-react-app with your actual DockerHub username and repository name.**

```
pipeline {

agent any

environment {

DOCKER_IMAGE = "Saravanave/trend-react-app"

DOCKERHUB_CREDENTIALS = "dockerhub_credentials"

// Update the path to the new location

KUBECONFIG_PATH = "/var/lib/jenkins/.kube/config"
```

```
}
stages {
stage('Git Checkout') {
steps {
// This is handled automatically by the Git plugin
echo 'Checking out code from Git...'
checkout scm
}
}
stage('Build Docker Image') {
steps {
script {
// Get the current Git commit hash for tagging the image
def gitHash = sh(returnStdout: true, script: 'git rev-parse --short HEAD').trim()
// Build the Docker image with the Git commit hash as a tag
sh "docker build -t ${DOCKER_IMAGE}:${gitHash} -t ${DOCKER_IMAGE}:latest ."
}
}
}
stage('Push to DockerHub') {
steps {
withCredentials([usernamePassword(credentialsId: DOCKERHUB_CREDENTIALS,
usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
script {
// Login to DockerHub
```

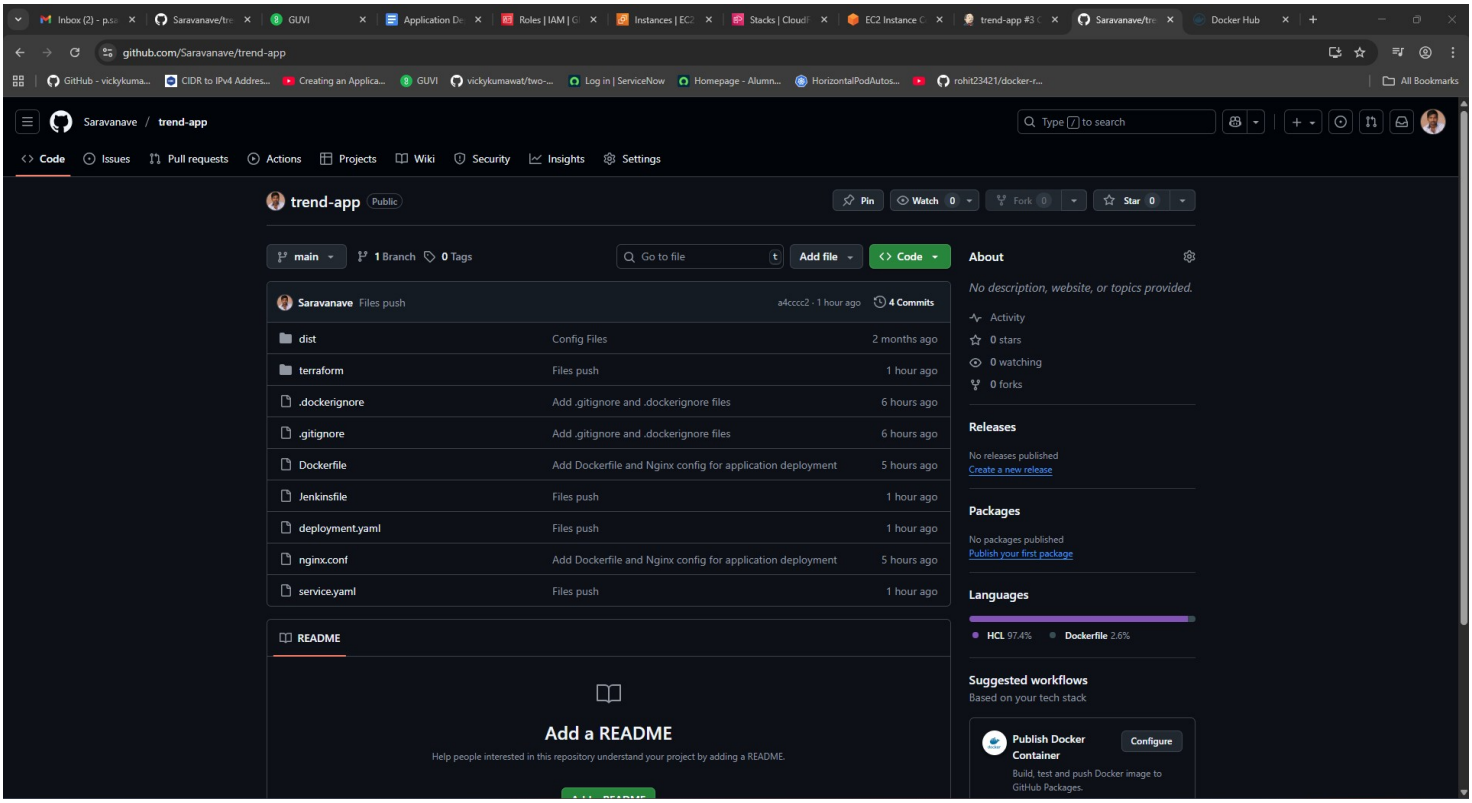
```
sh "echo $DOCKER_PASS | docker login --username $DOCKER_USER --passwordstdin"
// Push the 'latest' and commit-tagged images to DockerHub
sh "docker push ${DOCKER_IMAGE}:latest"
sh "docker push ${DOCKER_IMAGE}:${sh(returnStdout: true, script: 'git rev-parse --
short HEAD').trim()}"
}
}
}
}
stage('Deploy to Kubernetes') {
steps {
sh """
# Configure kubectl to use the kubeconfig file on the Jenkins server
export KUBECONFIG=/var/lib/jenkins/.kube/config
# Apply the Kubernetes Deployment and Service manifests
echo "Applying Kubernetes Deployment..."
kubectl apply -f deployment.yaml
echo "Applying Kubernetes Service..."
kubectl apply -f service.yaml
"""
}
}
}
}
```

4. Commit and Push the Jenkinsfile:

git add Jenkinsfile

git commit -m "Add Jenkinsfile for CI/CD pipeline"

git push origin main



8.2. Create the Pipeline Job in Jenkins

8.2.1. Access your Jenkins UI: http://<YOUR_JENKINS_PUBLIC_IP>:8080

8.2.2. Create a New Item:

- Click New Item on the left-hand side.
- Enter an item name: Trend-App-CI-CD
- Select a type: Pipeline
- Click OK.

8.2.3. Configure the Pipeline:

1. GitHub project: Enter your GitHub repository URL:

   https://github.com/Saravanave/trend-app.git

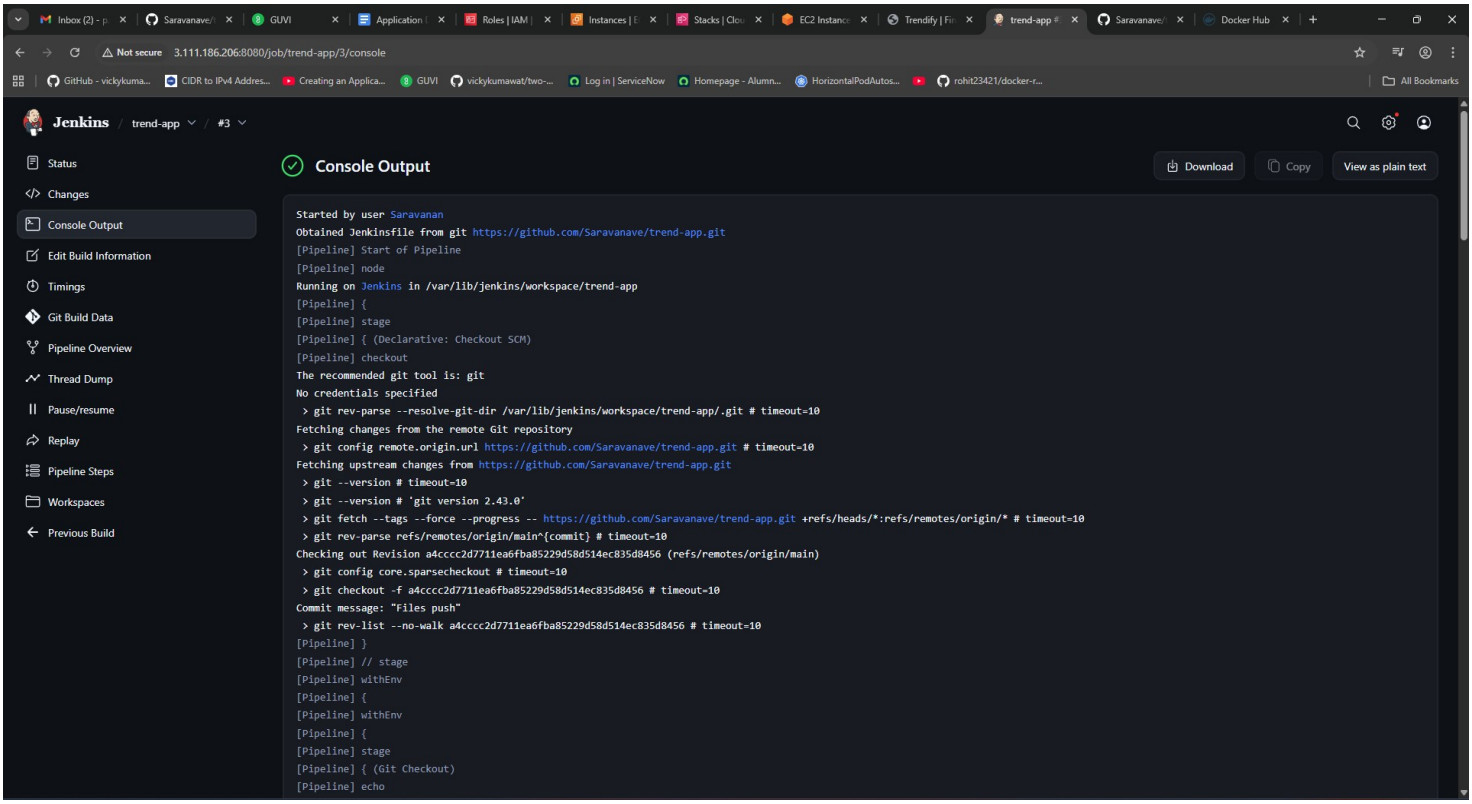
2. Build Triggers: Check the GitHub hook trigger for GITScm polling box. This ensures a build is triggered on every Git push.
3. Pipeline:

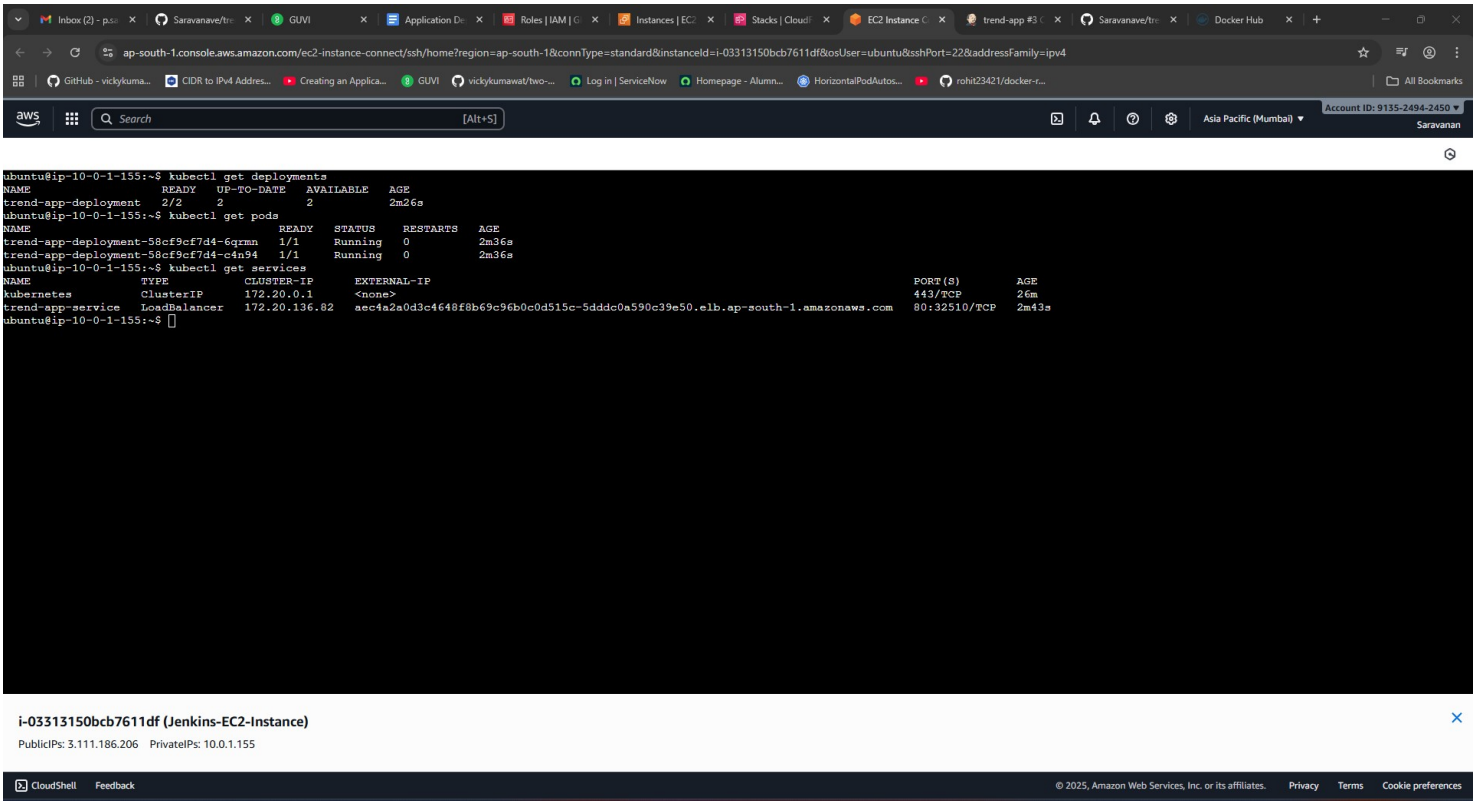
- ❖ Definition: Select Pipeline script from SCM.
- ❖ SCM: Select Git.
- ❖ Repository URL: Enter your GitHub repository URL:
- ❖ https://github.com/Saravanave/trend-app.git
- ❖ Credentials: Select - none - as your repository is public.
- ❖ Branch Specifier: main
- ❖ Script Path: Jenkinsfile

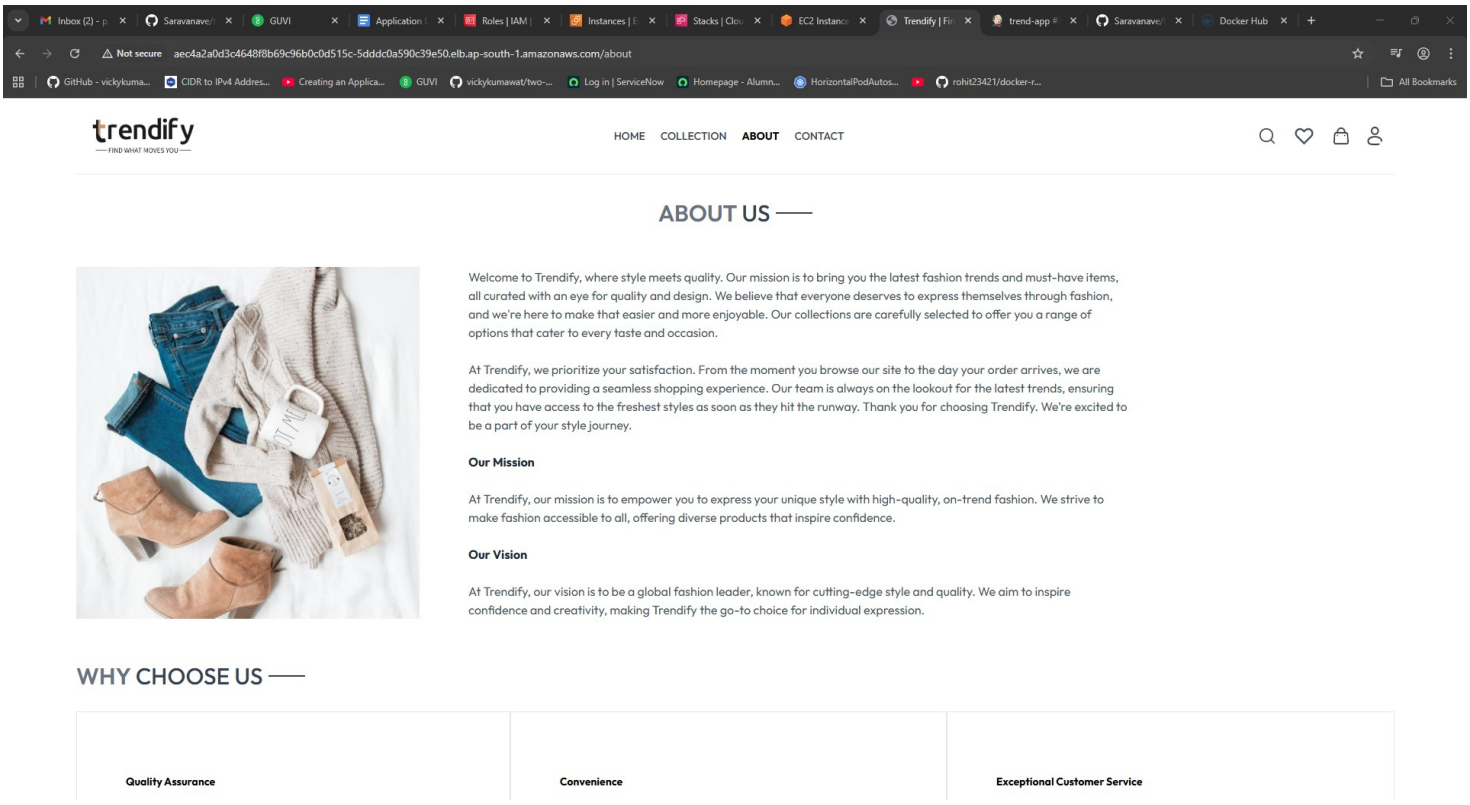
8.2.4.Save the job.

8.3. Trigger and Verify the Pipeline

- ❖ Go to your Jenkins job dashboard and click Build Now on the left.
- ❖ Wait for the build to complete. You can click on the build number to see the console output and watch each stage run.
- ❖ Once the pipeline completes successfully, go back to your SSH session on the Jenkins EC2 instance.
- ❖ Run kubectl get services and wait a few minutes for the EXTERNAL-IP (DNS name) of your service
- ❖ to be assigned. Copy the DNS name and paste it into your web browser. This time, you should see your application running!




ubuntu@ip-10-0-1-155:~$ kubectl get deployments
NAME READY UP-TO-DATE AVAILABLE AGE
trend-app-deployment 2/2 2 2 2m26s
ubuntu@ip-10-0-1-155:~$ kubectl get pods
NAME READY STATUS RESTARTS AGE
trend-app-deployment-58cf9cf7d4-6qrmn 1/1 Running 0 2m36s
trend-app-deployment-58cf9cf7d4-c4n94 1/1 Running 0 2m36s
ubuntu@ip-10-0-1-155:~$ kubectl get services
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 172.20.0.1 <none> 443/TCP 26m
trend-app-service LoadBalancer 172.20.136.82 aec4a2a0d3c4648f8b69c96b0c0d515c-5dddc0a590c39e50.elb.ap-south-1.amazonaws.com 80:32510/TCP 2m43s
ubuntu@ip-10-0-1-155:~$
i-03313150bcb7611df (Jenkins-EC2-Instance)
PublicIPs: 3.111.186.206 PrivateIPs: 10.0.1.155
CloudShell Feedback
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences


Not secure aec4a2a0d3c4648f8b69c96b0c0d515c-5dddc0a590c39e50.elb.ap-south-1.amazonaws.com/about
trendify
FIND WHAT MOVES YOU
HOME COLLECTION ABOUT CONTACT
ABOUT US
Welcome to Trendify, where style meets quality. Our mission is to bring you the latest fashion trends and must-have items, all curated with an eye for quality and design. We believe that everyone deserves to express themselves through fashion, and we're here to make that easier and more enjoyable. Our collections are carefully selected to offer you a range of options that cater to every taste and occasion.
At Trendify, we prioritize your satisfaction. From the moment you browse our site to the day your order arrives, we are dedicated to providing a seamless shopping experience. Our team is always on the lookout for the latest trends, ensuring that you have access to the freshest styles as soon as they hit the runway. Thank you for choosing Trendify. We're excited to be a part of your style journey.
Our Mission
At Trendify, our mission is to empower you to express your unique style with high-quality, on-trend fashion. We strive to make fashion accessible to all, offering diverse products that inspire confidence.
Our Vision
At Trendify, our vision is to be a global fashion leader, known for cutting-edge style and quality. We aim to inspire confidence and creativity, making Trendify the go-to choice for individual expression.
WHY CHOOSE US
Quality Assurance
Convenience
Exceptional Customer Service

Application deployed kubernetes Loadbalancer ARN:

http://aec4a2a0d3c4648f8b69c96b0c0d515c-5dddc0a590c39e50.elb.ap-south-1.amazonaws.com/

Step 9: Set up Prometheus and Grafana for Monitoring

9.1. Install Helm

First, you need to install Helm on your Jenkins EC2 instance if you haven't already.

1. SSH into your Jenkins EC2 instance.

2. Run the following commands to install Helm:

curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

9.2. Install the Prometheus Stack with Helm

We'll use a Helm chart called kube-prometheus-stack, which is a comprehensive bundle that includes Prometheus, Grafana, and all the necessary components for Kubernetes monitoring.

1. Add the Prometheus Helm repository:

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm repo update

2. Next, create a file named grafana-values.yaml on your Jenkins instance with the following content. This file tells Helm to create a public Load Balancer for Grafana.

grafana-values.yaml

grafana:

service:

type: LoadBalancer

port: 80

targetPort: 3000

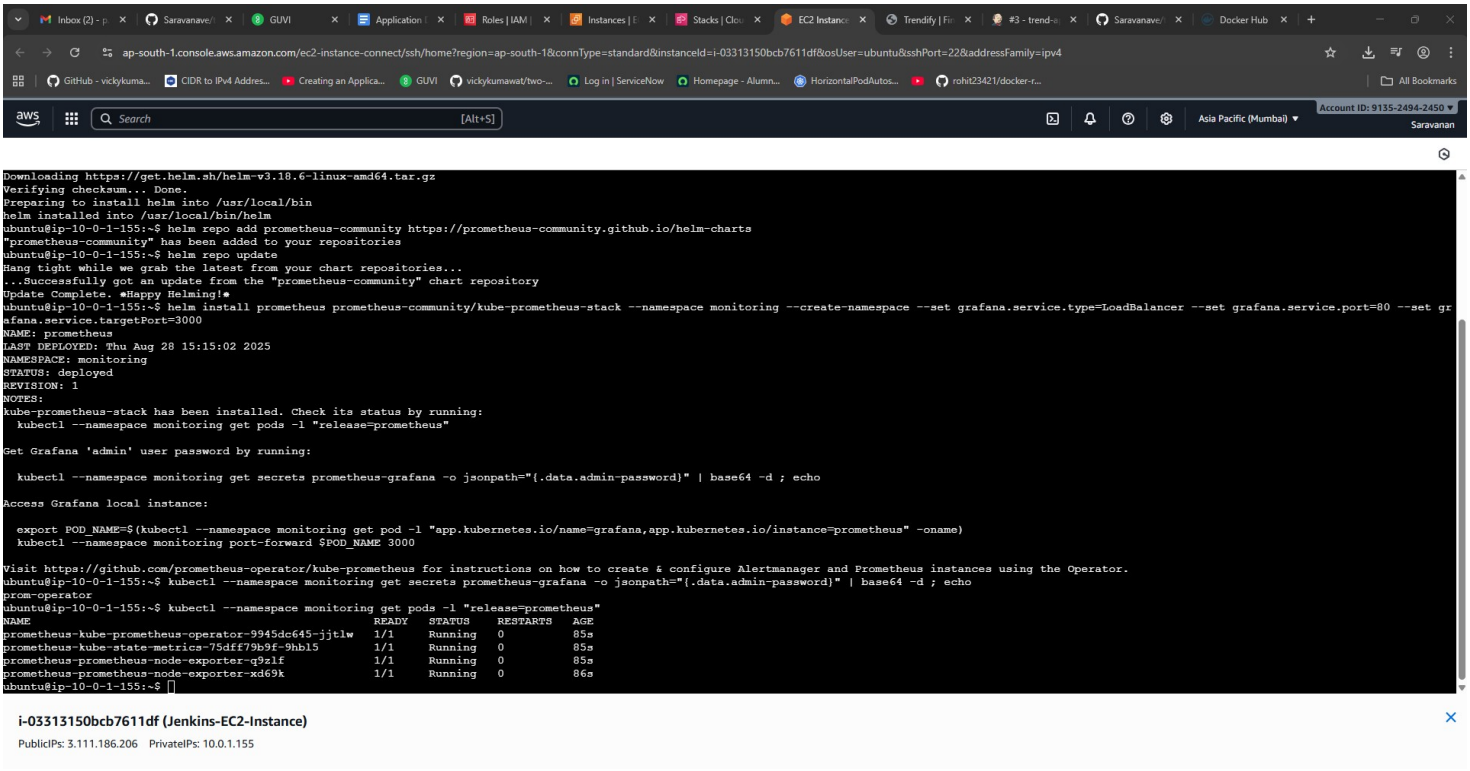
3. Install the kube-prometheus-stack chart. This will deploy Prometheus and Grafana to a new namespace called monitoring.

helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring --

create-namespace

4. Verify that the pods are running in the monitoring namespace:

kubectl get pods -n monitoring



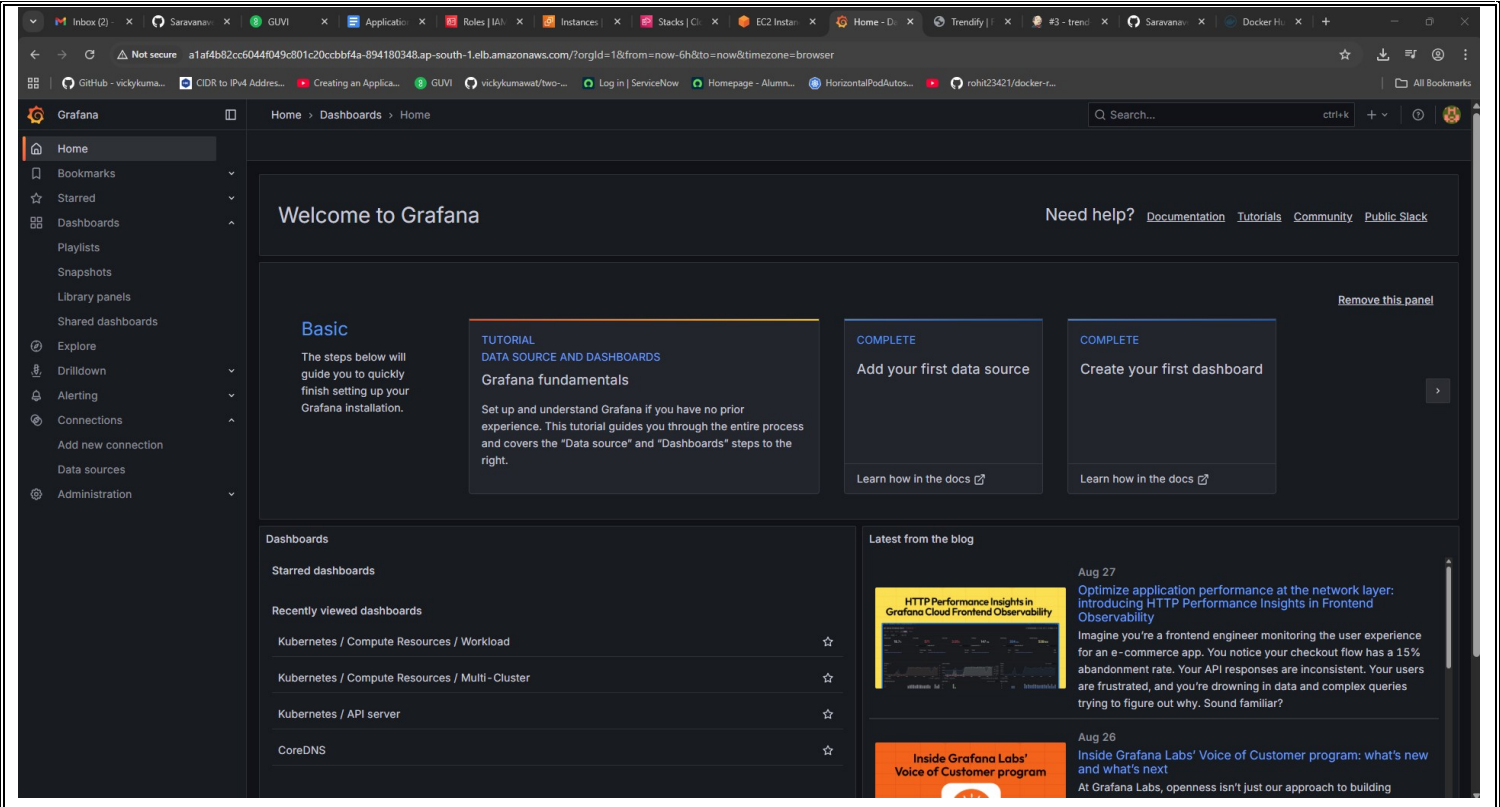
Look for the prometheus-grafana service. It will now have an AWS DNS name listed under the

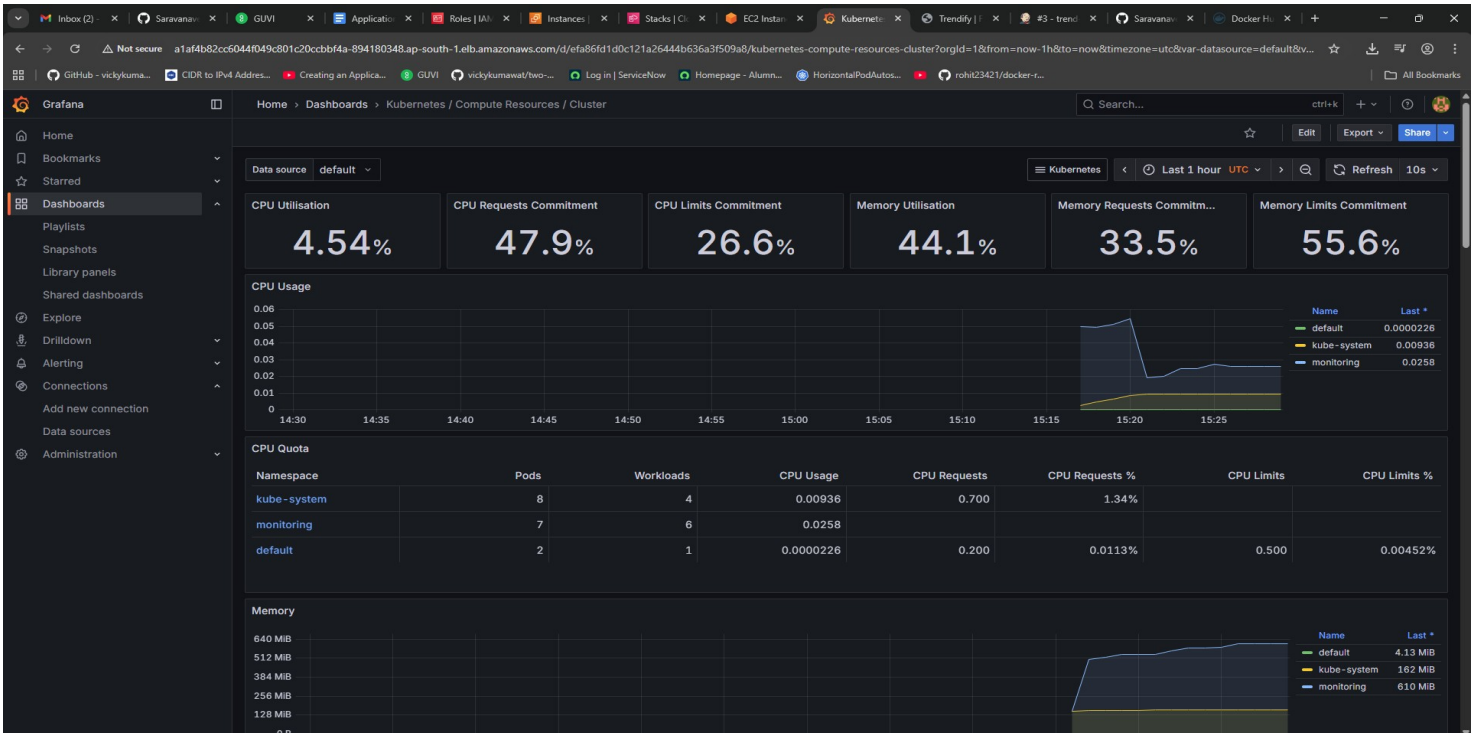
EXTERNAL-IP column. Copy this DNS name, paste it into your browser, and you should be able to see the grafana login page.

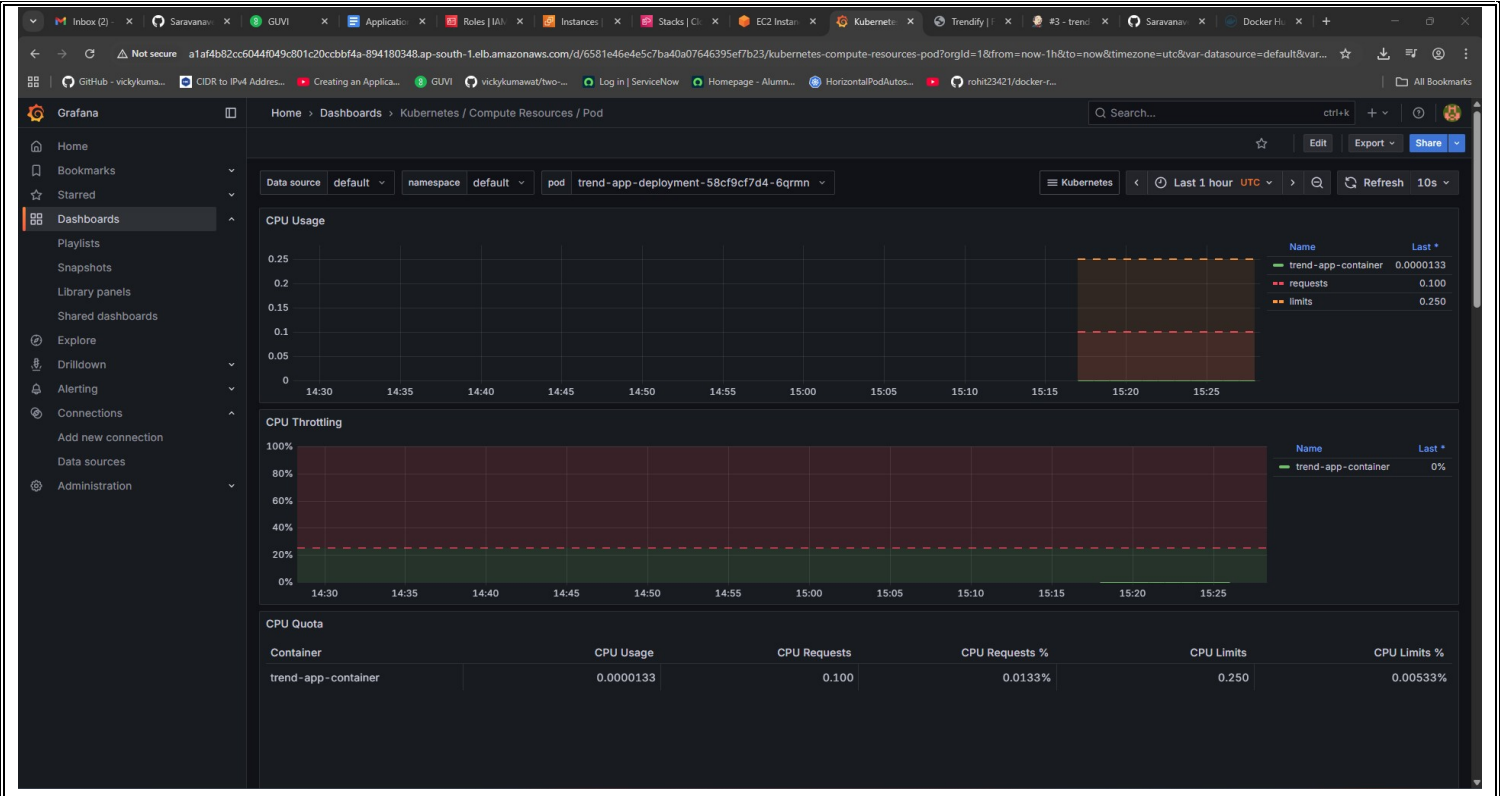
Default Credentials:

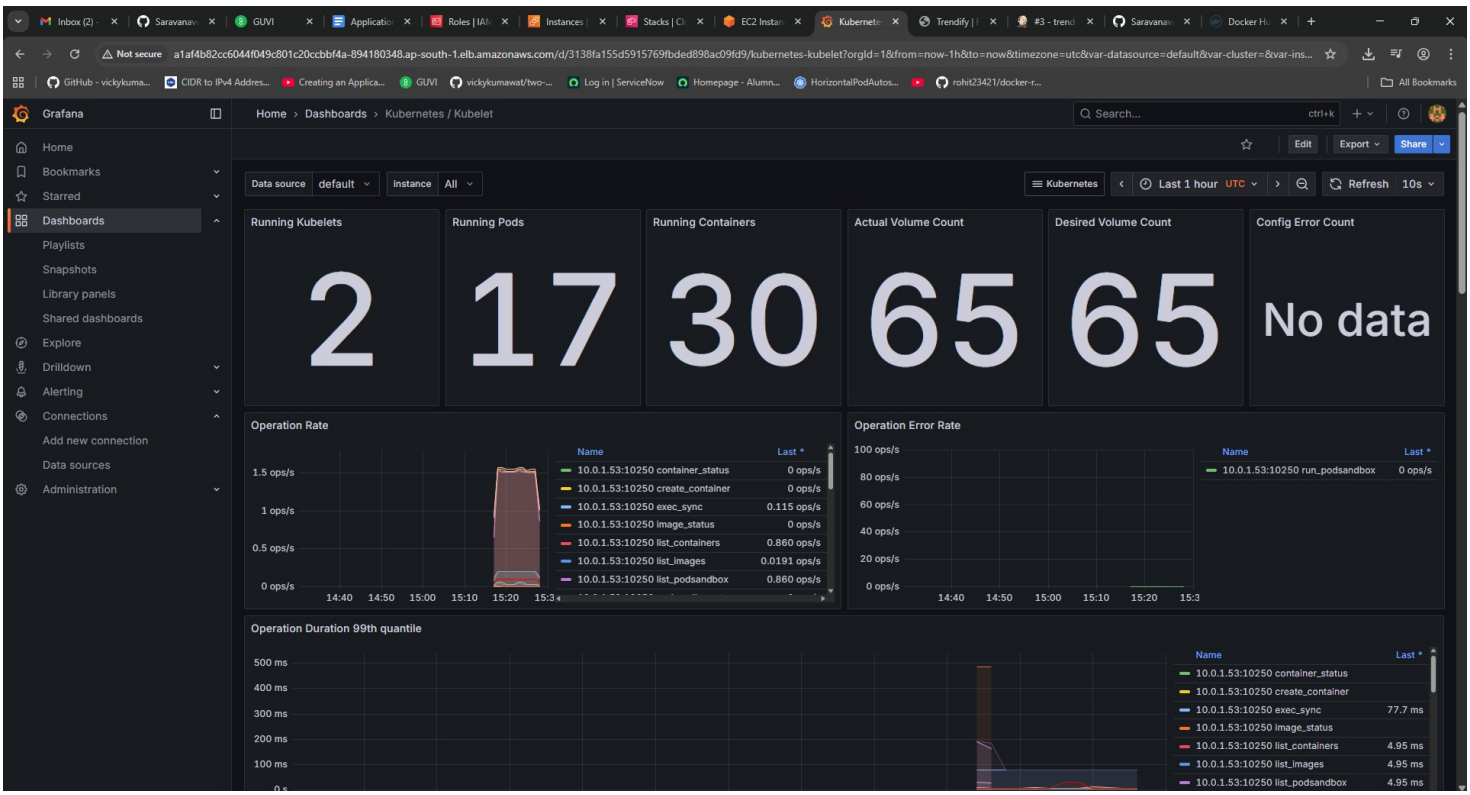
Username: admin

Password: prom-operator


Not secure a1af4b82cc6044f049c801c20ccbbf4a-894180348.ap-south-1.elb.amazonaws.com/?orgId=1&from=now-6h&to=now&timezone=browser
Grafana
Home › Dashboards › Home
Search... ctrl+k
Home
Bookmarks
Starred
Dashboards
Playlists
Snapshots
Library panels
Shared dashboards
Explore
Drilldown
Alerting
Connections
Add new connection
Data sources
Administration
Welcome to Grafana
Need help? Documentation Tutorials Community Public Slack
Remove this panel
Basic
The steps below will guide you to quickly finish setting up your Grafana installation.
TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals
Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.
COMPLETE
Add your first data source
Learn how in the docs
COMPLETE
Create your first dashboard
Learn how in the docs
Dashboards
Starred dashboards
Recently viewed dashboards
Kubernetes / Compute Resources / Workload
Kubernetes / Compute Resources / Multi-Cluster
Kubernetes / API server
CoreDNS
Latest from the blog
HTTP Performance Insights in Grafana Cloud Frontend Observability
Aug 27
Optimize application performance at the network layer: introducing HTTP Performance Insights in Frontend Observability
Imagine you're a frontend engineer monitoring the user experience for an e-commerce app. You notice your checkout flow has a 15% abandonment rate. Your API responses are inconsistent. Your users are frustrated, and you're drowning in data and complex queries trying to figure out why. Sound familiar?
Inside Grafana Labs' Voice of Customer program
Aug 26
Inside Grafana Labs' Voice of Customer program: what's new and what's next
At Grafana Labs, openness isn't just our approach to building


Not secure a1af4b82cc6044f049c801c20ccbbf4a-894180348.ap-south-1.elb.amazonaws.com/d/efa86fd1d0c121a26444b636a3f509a8/kubernetes-compute-resources-cluster?orgId=1&from=now-1h&to=now&timezone=utc&var-datasource=default&v...
Grafana
Home › Dashboards › Kubernetes / Compute Resources / Cluster
Search... ctrl+k
Edit
Export
Share
Data source default
Kubernetes
Last 1 hour UTC
Refresh 10s
CPU Utilisation
4.54%
CPU Requests Commitment
47.9%
CPU Limits Commitment
26.6%
Memory Utilisation
44.1%
Memory Requests Commitm...
33.5%
Memory Limits Commitment
55.6%
CPU Usage
0.06
0.05
0.04
0.03
0.02
0.01
0
14:30 14:35 14:40 14:45 14:50 14:55 15:00 15:05 15:10 15:15 15:20 15:25
Name Last *
default 0.0000226
kube-system 0.00936
monitoring 0.0258
CPU Quota
Namespace Pods Workloads CPU Usage CPU Requests CPU Requests % CPU Limits CPU Limits %
kube-system 8 4 0.00936 0.700 1.34%
monitoring 7 6 0.0258
default 2 1 0.0000226 0.200 0.0113% 0.500 0.00452%
Memory
640 MiB
512 MiB
384 MiB
256 MiB
128 MiB
Name Last *
default 4.13 MiB
kube-system 162 MiB
monitoring 610 MiB


Grafana
Home › Dashboards › Kubernetes / Compute Resources / Pod
Search...
Edit
Export
Share
Data source default
namespace default
pod trend-app-deployment-58cf9cf7d4-6qrmn
Kubernetes
Last 1 hour UTC
Refresh 10s
CPU Usage
Name Last *
trend-app-container 0.0000133
requests 0.100
limits 0.250
CPU Throttling
Name Last *
trend-app-container 0%
CPU Quota
Container CPU Usage CPU Requests CPU Requests % CPU Limits CPU Limits %
trend-app-container 0.0000133 0.100 0.0133% 0.250 0.00533%


Grafana
Home › Dashboards › Kubernetes / Kubelet
Search...
Edit
Export
Share
Data source default
instance All
Kubernetes
Last 1 hour UTC
Refresh 10s
Running Kubelets
2
Running Pods
17
Running Containers
30
Actual Volume Count
65
Desired Volume Count
65
Config Error Count
No data
Operation Rate
Name Last *
10.0.1.53:10250 container_status 0 ops/s
10.0.1.53:10250 create_container 0 ops/s
10.0.1.53:10250 exec_sync 0.115 ops/s
10.0.1.53:10250 image_status 0 ops/s
10.0.1.53:10250 list_containers 0.860 ops/s
10.0.1.53:10250 list_images 0.0191 ops/s
10.0.1.53:10250 list_podsandbox 0.860 ops/s
Operation Error Rate
Name Last *
10.0.1.53:10250 run_podsandbox 0 ops/s
Operation Duration 99th quantile
Name Last *
10.0.1.53:10250 container_status
10.0.1.53:10250 create_container
10.0.1.53:10250 exec_sync 77.7 ms
10.0.1.53:10250 image_status
10.0.1.53:10250 list_containers 4.95 ms
10.0.1.53:10250 list_images 4.95 ms
10.0.1.53:10250 list_podsandbox 4.95 ms