KUBERNETES DEPLOYMENT PIPELINE FOR FEEDBACK APP WITH JENKINS AND KOPS

Github to Kubernetes

Ramnath P
Ramnathraja935@gmail.com
+91 9791438315

Table Of Content

Table Of Content

1. Abstract:

This project aims to streamline the deployment process of a GitHub project onto a Kubernetes server, facilitating seamless access to the application for users and developers alike. In today's fast-paced software development landscape, efficient deployment is crucial for maintaining competitive advantage and ensuring quick delivery of features and updates. By leveraging the powerful capabilities of Kubernetes for container orchestration, we provide a robust framework that automates the deployment, scaling, and management of applications within a cloud environment.

The project encompasses the comprehensive setup of a continuous integration and continuous deployment (CI/CD) pipeline, which is essential for automating the process of integrating code changes and deploying them to the Kubernetes server. This pipeline ensures that updates to the GitHub repository are efficiently reflected in the Kubernetes deployment, reducing manual intervention and the potential for errors.

In addition to deployment automation, we will also implement monitoring and logging solutions to track application performance and facilitate troubleshooting. Through this initiative, we aim to enhance the reliability, scalability, and accessibility of applications, ultimately driving successful project outcomes. By enabling a more agile and responsive deployment process, this project aspires to empower teams to innovate and deliver value to their users swiftly and efficiently.

2. Virtual Private Cloud:

A Virtual Private Cloud (VPC) is a secure, isolated section of a cloud provider's network where users can launch resources in a virtualized environment. It allows for customizable network configurations, including subnets, IP address ranges, and routing tables, while enabling secure communication between resources. VPCs enhance security by providing private IP addresses and the ability to define access controls through security groups and network ACLs. They are ideal for deploying applications that require a secure and controlled networking environment. Used the default VPC for the instance and eksctl-my-eks-cluster-cluster/VPC for the eks cluster.

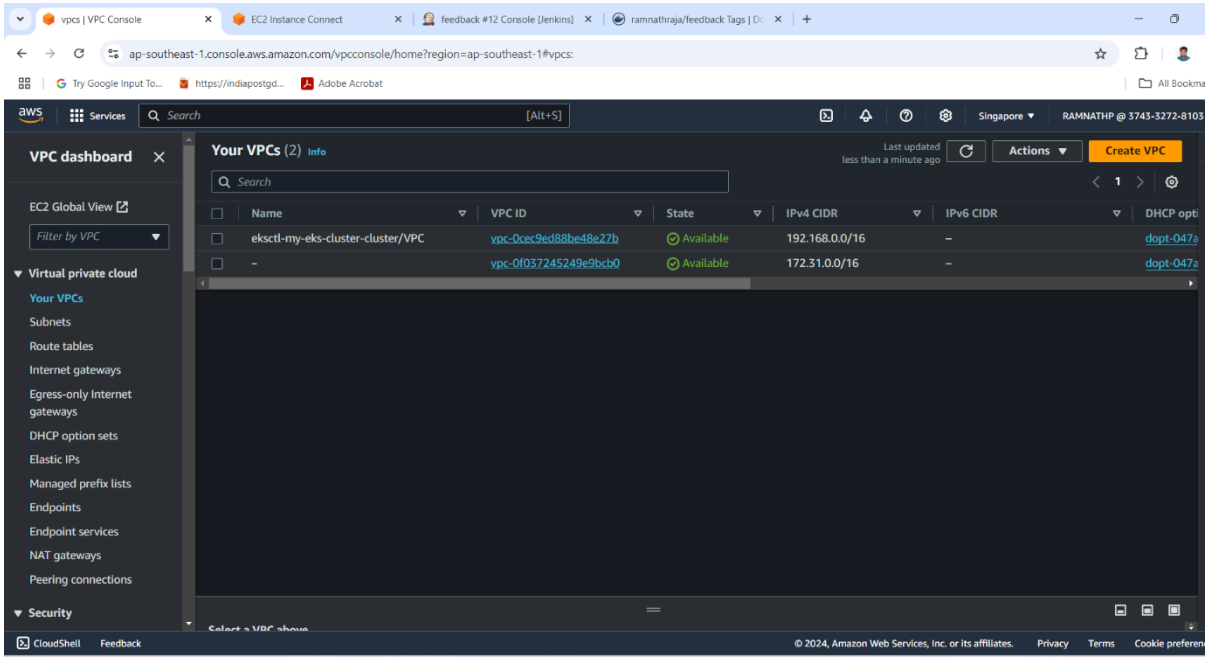

Fig 1. VPC of the project

3. Subnets:

A subnet (subnetwork) is a smaller, segmented part of a larger network. It divides the network into manageable sections, each with its own range of IP addresses. Subnets improve network performance by reducing congestion, enhance security through isolation, and simplify management. They enable efficient routing and can contain broadcast traffic, making networks more organized and efficient. Used the default Subnet for the instance and the following subnets for eks cluster,

eksctl-my-eks-cluster-cluster/SubnetPrivateAPSOUTHEAST1B

eksctl-my-eks-cluster-cluster/SubnetPublicAPSOUTHEAST1A

eksctl-my-eks-cluster-cluster/SubnetPublicAPSOUTHEAST1C

eksctl-my-eks-cluster-cluster/SubnetPublicAPSOUTHEAST1B

eksctl-my-eks-cluster-cluster/SubnetPrivateAPSOUTHEAST1A

eksctl-my-eks-cluster-cluster/SubnetPrivateAPSOUTHEAST1C

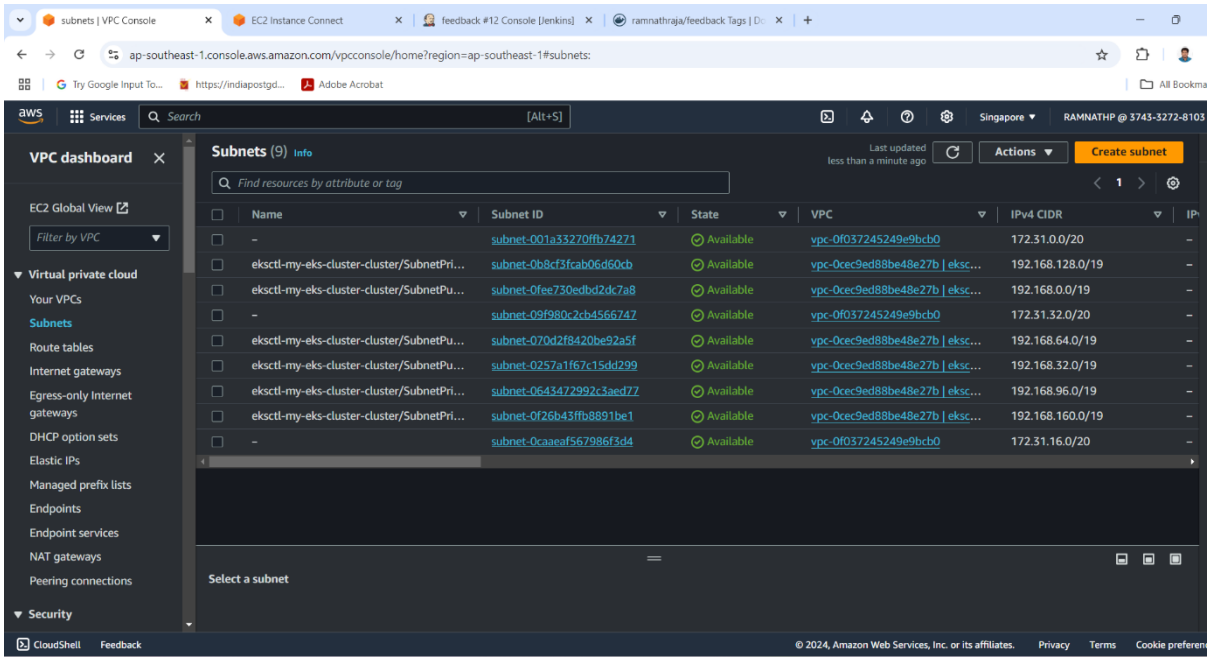

Fig 2. Subnet of the project

4. Route Table:

A route table is a set of rules used by routers to determine the best path for forwarding packets within a network. It contains entries that specify destination IP addresses, associated subnet masks, and the next hop or gateway for each route. Route tables help manage traffic, ensuring data is sent efficiently and correctly between different networks or subnets. They are essential for both local and internet routing, enabling effective communication between devices. The Subnets are connected to their respective route tables for instance and also for the eks cluster,

eksctl-my-eks-cluster-cluster/PrivateRouteTableAPSOUTHEAST1C

eksctl-my-eks-cluster-cluster/PrivateRouteTableAPSOUTHEAST1A

eksctl-my-eks-cluster-cluster/PublicRouteTable

eksctl-my-eks-cluster-cluster/PrivateRouteTableAPSOUTHEAST1B

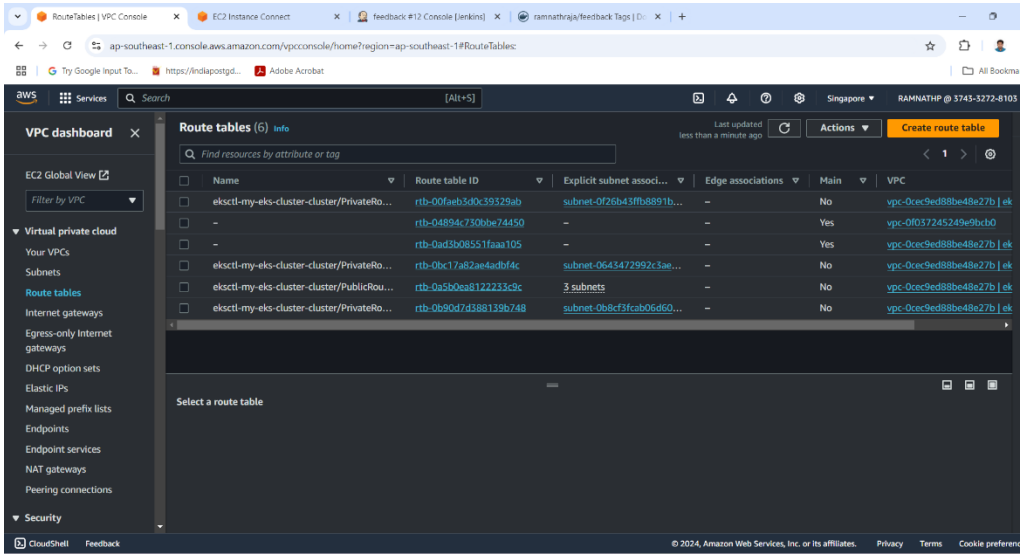

Fig 3. Route Table of the project

5. Internet Gateway:

An Internet Gateway is a horizontally scaled, redundant, and highly available component in a Virtual Private Cloud (VPC) that allows communication between instances in the VPC and the internet. It serves two main purposes:

Outbound Traffic: It enables resources in the VPC to access the internet.

Inbound Traffic: It allows external users to reach resources in the VPC, such as web servers, through public IP addresses.

The Internet Gateway ensures secure and efficient routing of traffic between the VPC and the internet.For the Internet Connectivity for the public subnets we have connected the Public Subnet where the public instances are connected to the Internet Gateway.

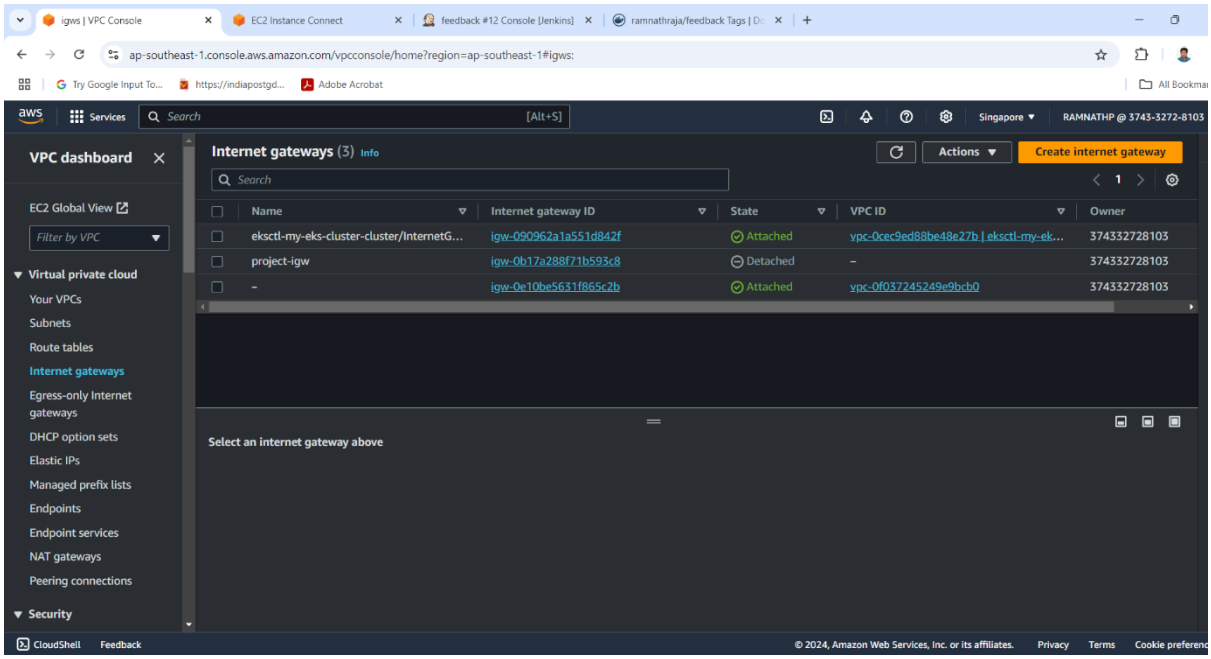

Fig 4. Internet Gateway of the project

6. NAT Gateway:

NAT (Network Address Translation) is a process that modifies the IP address information in packet headers while they are in transit across a router or firewall. It serves two main purposes:

IP Address Conservation: NAT allows multiple devices on a private network to share a single public IP address, helping conserve the limited number of available public IPs.

Security: By hiding internal IP addresses from external networks, NAT adds a layer of security, making it harder for attackers to reach devices on the private network.

In cloud environments, NAT Gateways facilitate outbound internet access for instances in private subnets while keeping them inaccessible from the internet. For the Internet Connectivity for the private subnets we have connected the private Subnet where the private instances are connected to the NAT(Network Address Translation) Gateway.

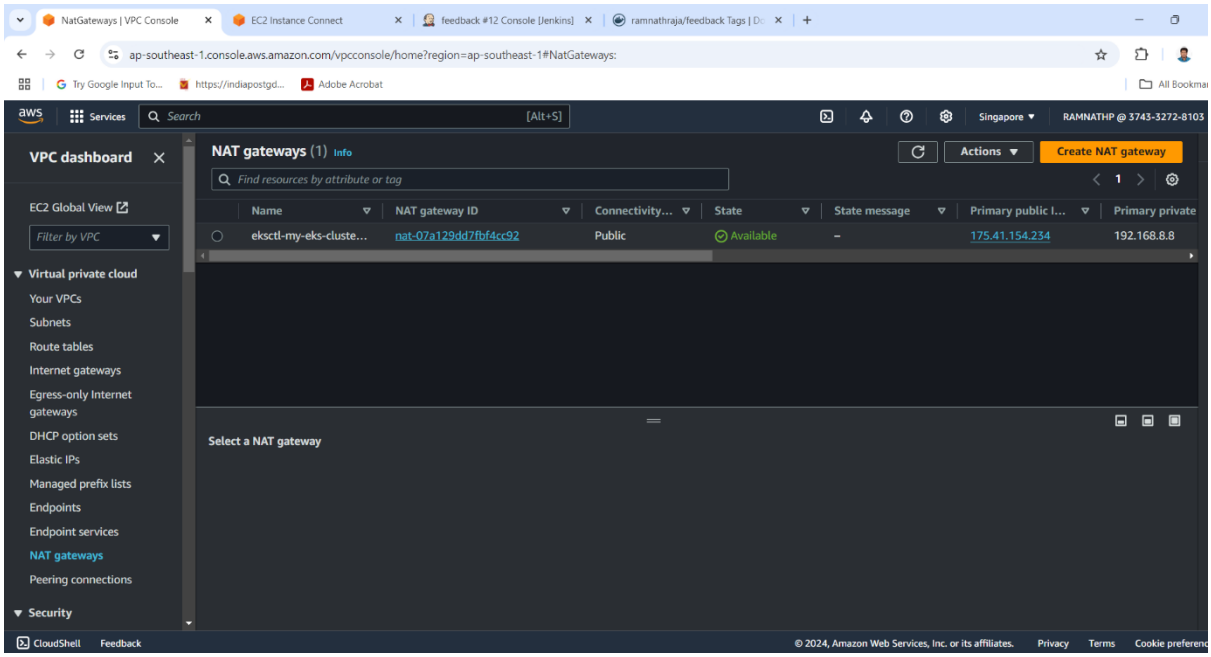

Fig 5. NAT Gateway of the project

7. Elastic IP:

An Elastic IP (EIP) is a static IP address designed for dynamic cloud computing, particularly used in Amazon Web Services (AWS). Here are the key points about Elastic Ips:

7.1. Static Nature: Unlike regular IP addresses that can change when an instance is stopped or terminated, an Elastic IP remains constant, providing a reliable endpoint for your applications.

7.2. Association: You can associate an Elastic IP with an AWS instance, allowing you to maintain the same IP address even if you stop or restart the instance.

7.3. Reassignment: Elastic Ips can be quickly reassigned to another instance in your account, making it easier to manage failover scenarios or instance migrations.

7.4. Cost: AWS charges for Elastic Ips if they are not associated with a running instance, so it's important to manage them efficiently.

7.5. Use Cases: Commonly used for web servers, databases, and other services where a consistent IP address is essential for communication.

This elastic IP is associated with the Cluster “my-eks-cluster”.

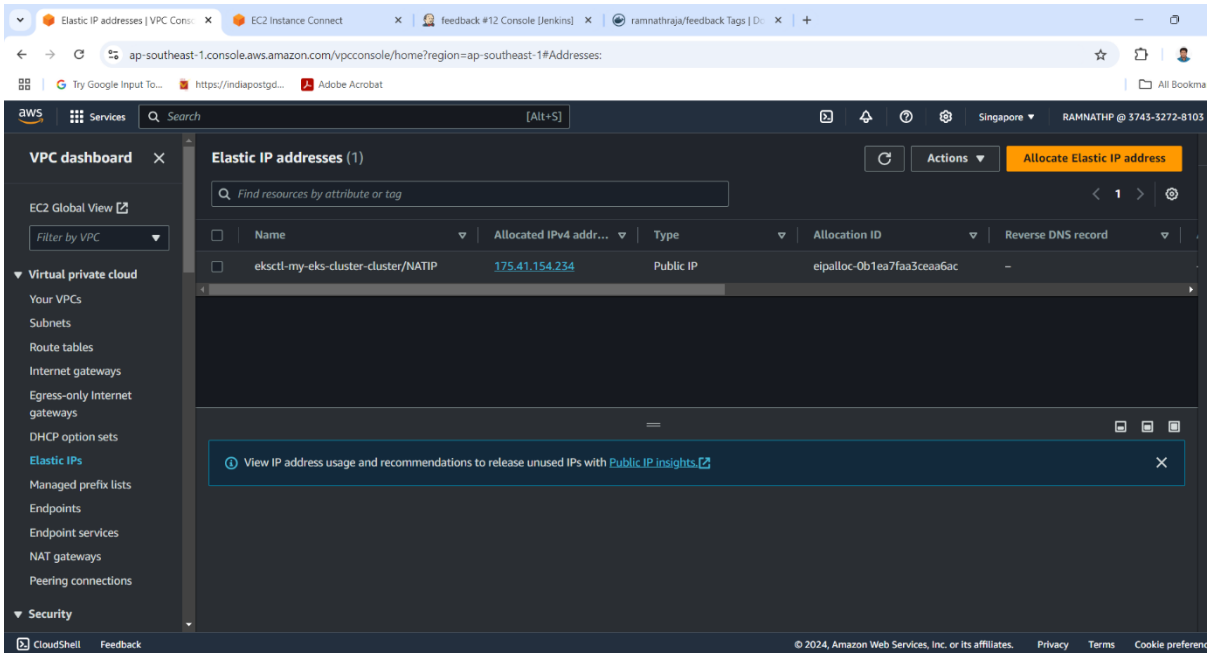

Fig 6. Elastic IP Address of the project

8. Github and Git:

Git and GitHub are closely related but serve different purposes in the world of software development.

8.1. Git:

Version Control System: Git is a distributed version control system used to track changes in source code during software development. It allows multiple developers to collaborate on a project by keeping track of file modifications and maintaining a history of changes.

8.2. Key Features:

Commit: Each change is saved as a "commit" with a unique identifier and message, enabling you to revert or track changes over time.

Branching and Merging: Git allows developers to create multiple branches to work on different features or fixes simultaneously. Branches can be merged back together once work is complete.

Distributed: Each user has a full copy of the repository on their machine, so Git works offline and is highly resilient.

8.3. Example Commands:

git init: Initializes a new Git repository.

git add: Stages files for a commit.

git commit: Commits the staged changes with a message.

git push: Pushes local changes to a remote repository.

git pull: Fetches and merges changes from a remote repository.

8.4. GitHub:

Hosting Service: GitHub is a cloud-based platform that hosts Git repositories. It provides a web-based interface to manage Git projects and offers collaboration tools for teams, such as issue tracking, pull requests, and project management features.

8.5. Key Features:

8.5.1. Remote Repositories: GitHub allows you to host your Git repositories in the cloud, enabling collaboration from anywhere.

8.5.2. Pull Requests: Developers can propose changes to a project by submitting a pull request, where changes can be reviewed, discussed, and approved before merging.

8.5.3. Forking: GitHub allows users to fork (copy) a repository and make changes independently, useful for open-source collaboration.

8.5.4. CI/CD Integration: GitHub integrates with continuous integration/continuous deployment (CI/CD) pipelines, automating testing and deployment of code.

This project files are stored in the github, the repository is named as feedback. And the link used is as follows,

HTTPS Link: https://github.com/ramnathraja/feedback.git

SSH Link: git@github.com:ramnathraja/feedback.git

Github CLI Link: gh repo clone ramnathraja/feedback

The project file are as follows,

Feedback-app-main

Dockerfile

Jenkinsfile

Deployment.yml

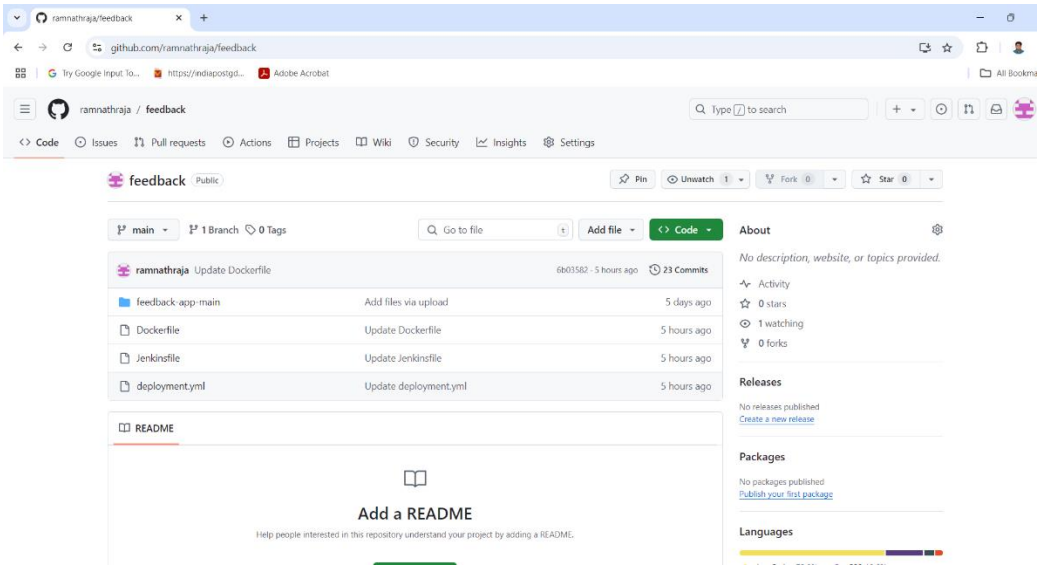

Fig 7. Github repo of the project

9. Installing Docker:

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run.

We can download the docker from the official docker website for the specified OS used in the EC2 instance. In this project we have used Ubuntu server hence I have used docker for ubuntu server.

The steps and commands are as follows:

Uninstall the old versions:

for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done

Installation using apt-repository

Set up Docker's apt repository:

sudo apt-get update

sudo apt-get install ca-certificates curl

sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

Add the repository to Apt sources:

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \

$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

Install the Docker packages:

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

we can verify the docker version using the command “docker –version” and also check whether any docker process is running using the command “docker ps”.

9.1. Dockerfile:

A **Dockerfile** is a script-like text file that contains a set of instructions to build a Docker image. It automates the process of creating Docker images by specifying the base image, dependencies, application code, configurations, and how the application should be executed. Here's a brief breakdown:

FROM: Specifies the base image to build upon. It could be an official image like ubuntu, node, or nginx

WORKDIR: Sets the working directory inside the container where subsequent commands will be executed.

COPY: Copies files from the host machine into the container.

RUN: Executes a command to install dependencies, run scripts, or perform actions during image build.

CMD: Defines the default command to run when the container starts. Unlike RUN, this is executed at runtime.

EXPOSE: Specifies the port on which the container listens at runtime.

The Dockerfile used in our project is as below,

```
# Use an official base image

FROM node:14

# Set the working directory inside the container

WORKDIR /app

# Copy package.json and package-lock.json for installing dependencies

COPY package*.json ./

# Install the dependencies

RUN npm install

# Copy all other application files

COPY . .

# Expose the port on which the app will run
```

```
EXPOSE 3000

# Command to start the application
CMD ["npm", "start"]
```

10. Installing the kubeadm:

Kubeadm is a tool built to provide kubeadm init and kubeadm join as best-practice "fast paths" for creating Kubernetes clusters.

kubeadm performs the actions necessary to get a minimum viable cluster up and running. By design, it cares only about bootstrapping, not about provisioning machines. Likewise, installing various nice-to-have addons, like the Kubernetes Dashboard, monitoring solutions, and cloud-specific addons, is not in scope.

Instead, we expect higher-level and more tailored tooling to be built on top of kubeadm, and ideally, using kubeadm as the basis of all deployments will make it easier to create conformant clusters.

10.1. The installation steps are as follows:

First we need to turn off the swap by using the command “sudo swapoff -a” and if we want to permanently off the swap we can hash the entry in the “/etc/fstab”.

Now for installing the container runtime go to the cri-dockerd in the note of the installation page, give getting started and click release-page and copy the cri-dockerd link of the bookworm and use the wget command and the https://github.com/Mirantis/cri-dockerd/releases/download/v0.3.15/cri-dockerd_0.3.15.3-0.debian-bookworm_amd64.deb to download the application file.

Give the executable permission to the file using the command chmod +x https://github.com/Mirantis/cri-dockerd/releases/download/v0.3.15/cri-dockerd_0.3.15.3-0.debian-bookworm_amd64.deb

Now install the application using the apt install https://github.com/Mirantis/cri-dockerd/releases/download/v0.3.15/cri-dockerd_0.3.15.3-0.debian-bookworm_amd64.deb

10.2. Installing kubeadm, kubelet and kubectl:

As I have used Debian/Ubuntu based machine I have followed the following Debian based distribution:

Update the apt package index and install packages needed to use the Kubernetes apt repository:

sudo apt-get update

 # apt-transport-https may be a dummy package; if so, you can skip that package

sudo apt-get install -y apt-transport-https ca-certificates curl gpg

Download the public signing key for the Kubernetes package repositories. The same signing key is used for all repositories so you can disregard the version in the URL:

 # If the directory `/etc/apt/keyrings` does not exist, it should be created before the curl command, read the note below.

 # sudo mkdir -p -m 755 /etc/apt/keyrings

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

Add the appropriate Kubernetes apt repository. Please note that this repository have packages only for Kubernetes 1.31; for other Kubernetes minor versions, you need to change the Kubernetes minor version in the URL to match your desired minor version (you should also check that you are reading the documentation for the version of Kubernetes that you plan to install).

 # This overwrites any existing configuration in /etc/apt/sources.list.d/kubernetes.list

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

Update the apt package index, install kubelet, kubeadm and kubectl, and pin their version:

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

Enable the kubelet service before running kubeadm:

sudo systemctl enable --now kubelet

10.3. Creating a cluster with kubeadm:

10.3.1. Installing a Pod network add-on:

Go to the Installing a Pod network add-on and click the calico network,click get started, Install Calico, Kubernetes, Quickstart for Calico on Kubernetes and put the following command

kubeadm init --pod-network-cidr=192.168.0.0/16 –cri-socket=unix:///var/run/cri-dockerd.sock

To make kubectl work for your non-root user, run these commands, which are also part of the kubeadm init output, put the following command

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown **$(**id -u**)**:**$(**id -g**)** $HOME/.kube/config

if you are the root user, we can run the following command,

export KUBECONFIG=/etc/kubernetes/admin.conf

We will get the joining token for joining the worker nodes to add into the master node cluster.

10.4. Download and install eksctl:

To download the eksctl to work on the EKS cluster in the Jenkins server CLI, follow the following commands,

curl -sL "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_Linux_amd64.tar.gz" -o eksctl.tar.gz

Extract the downloaded file using the command,

tar -xzf eksctl.tar.gz

Move the eksctl binary to /usr/local/bin using the command,

sudo mv eksctl /usr/local/bin

Creation of EKS Cluster using CLI:

We need to configure the aws credential using the command “aws configure” and mention the following details,

Aws access key ID, Secret Access key, region (default),output format

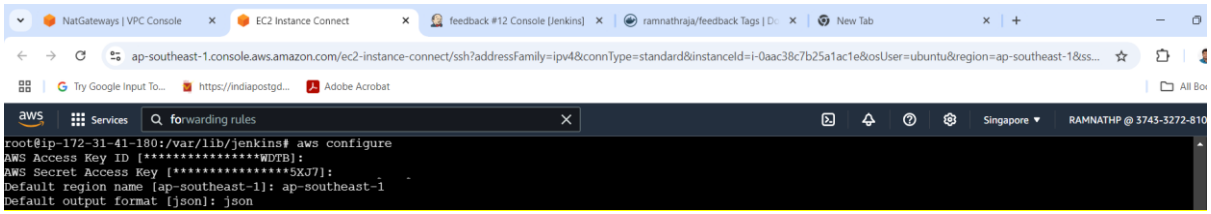

Fig 8. AWS configure command output

eksctl create cluster --name cluster name --region region name --nodegroup-name nodegroup name --nodes 2 --nodes-min 1 --nodes-max 2 –managed

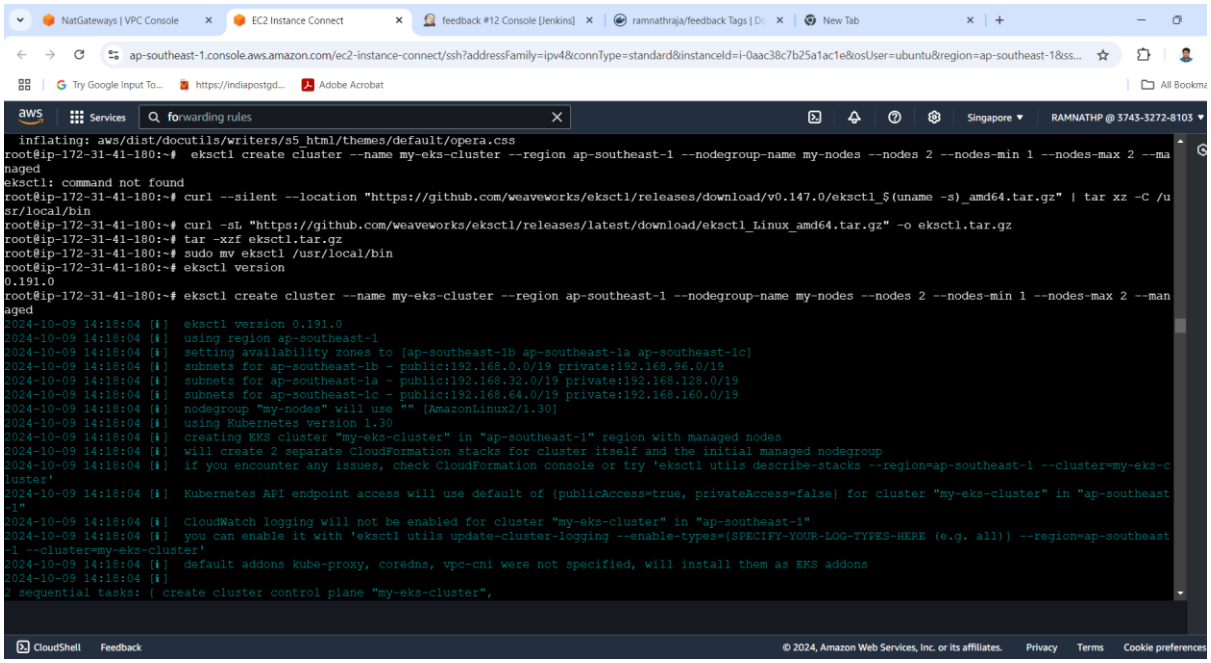

Fig 9.1. Creation of EKS Cluster 1

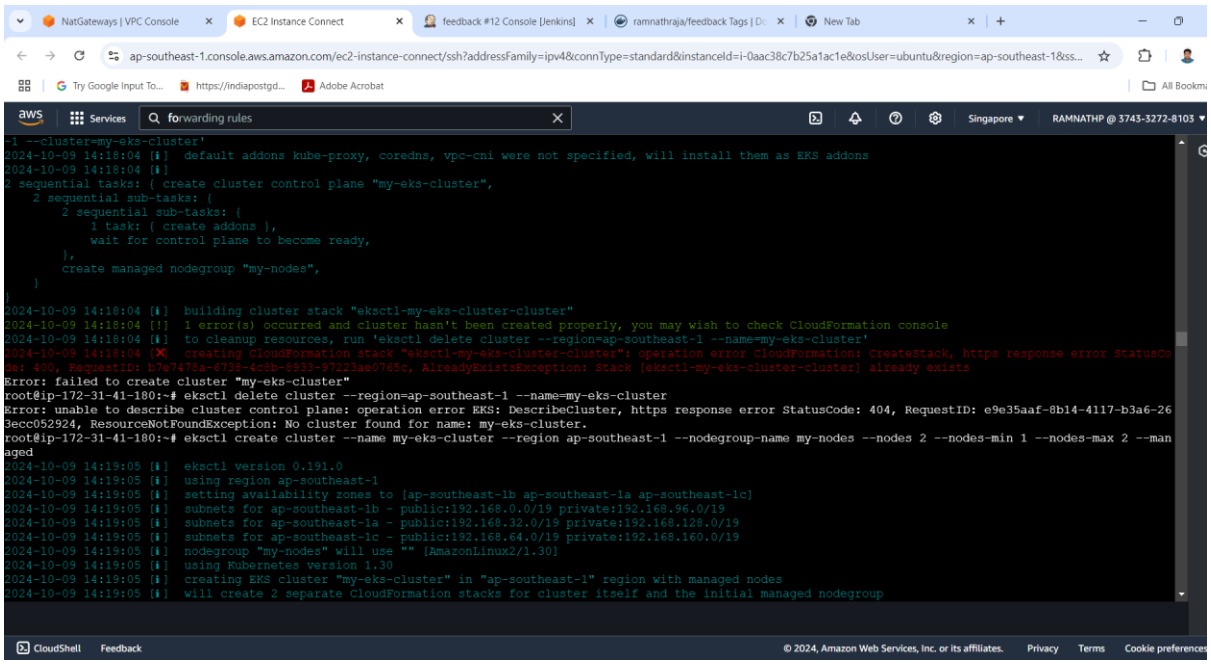

Fig 9.2. Creation of EKS Cluster 2

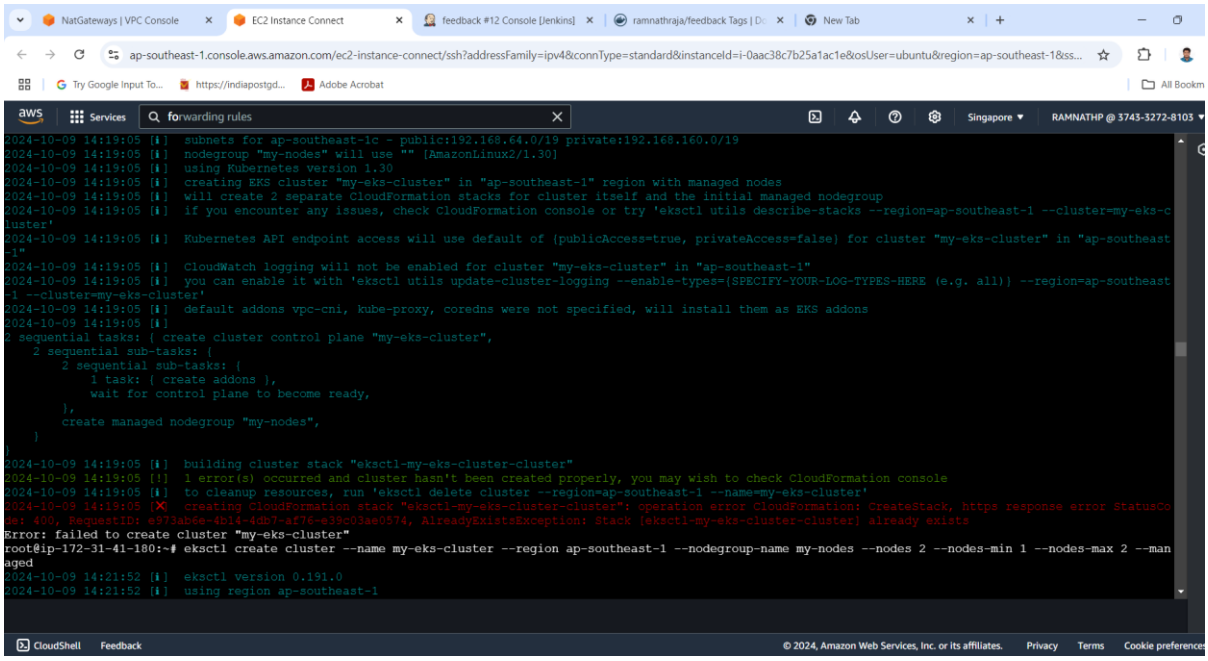

Fig 9.3. Creation of EKS Cluster 3

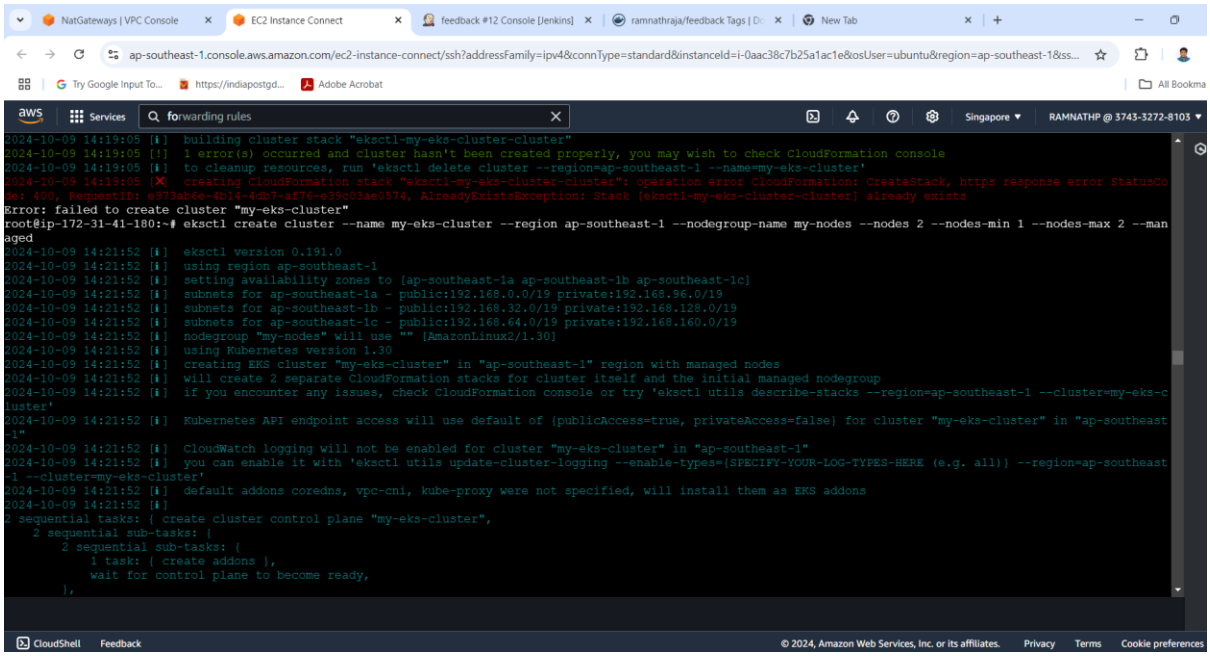

Fig 9.4. Creation of EKS Cluster(Correct output 1)

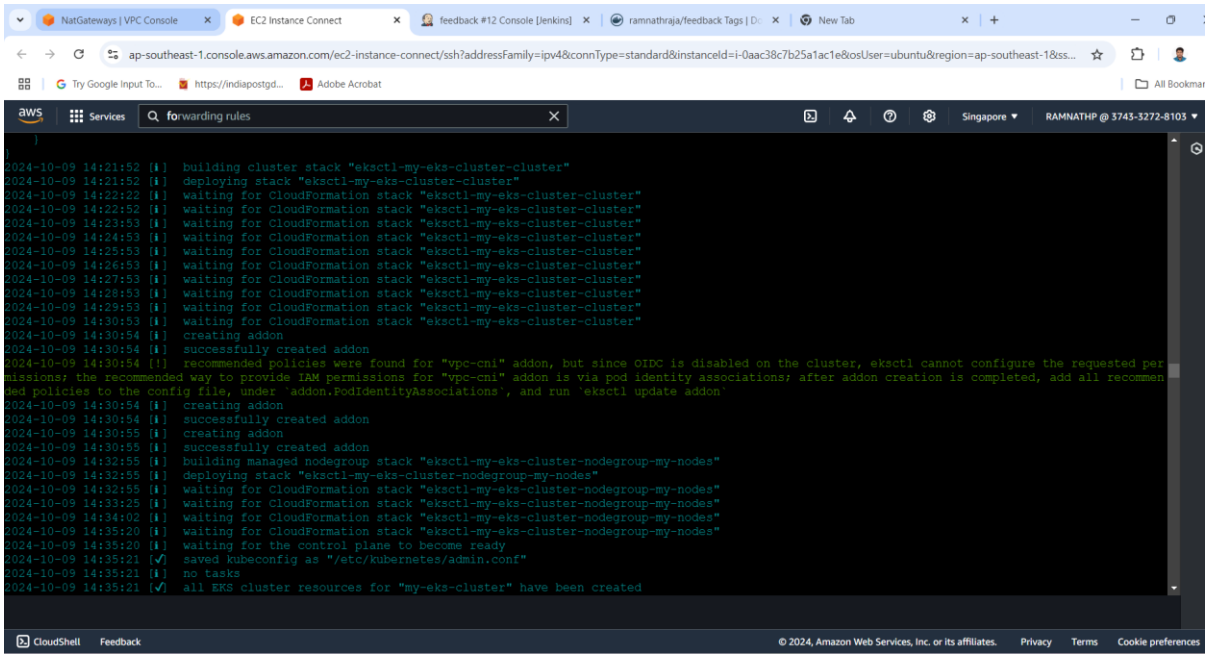

Fig 9.5. Creation of EKS Cluster(Correct output 2)

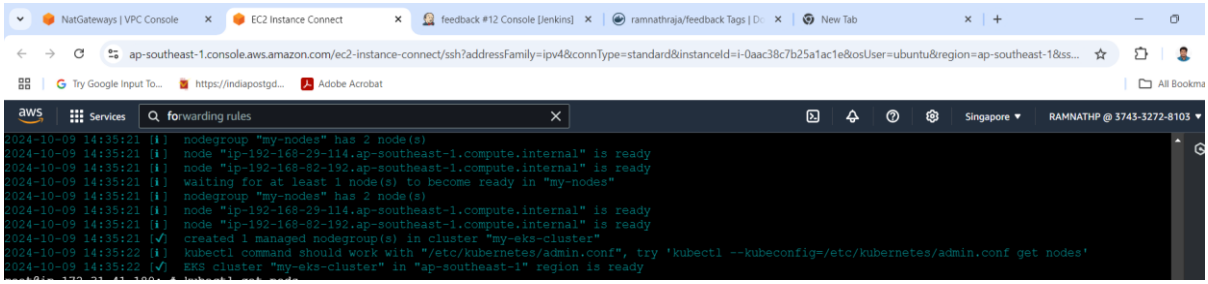

Fig 9.6. Creation of EKS Cluster(Correct output 3)

10.5. Issue and Workaround:

When giving aws configure command we can get the aws command not found error so we need to add the aws repo and then install the awscli using the following command,

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

unzip awscliv2.zip

sudo ./aws/install

And sometimes we can also get unzip command not found in that case use the below command,

apt install unzip

11. Installing Jenkins:

Jenkins is an open-source automation server widely used for continuous integration and continuous delivery (CI/CD). It helps automate the building, testing, and deployment of software projects, allowing developers to integrate code changes more frequently and efficiently.

11.1. Key Features:

11.1.1. Extensibility: Jenkins has a rich ecosystem of plugins that allow integration with various tools and technologies.

11.1.2. Distributed Builds: It can distribute tasks across multiple machines to speed up the build process.

11.1.3. Pipeline Support: Jenkins supports defining build processes as code through Jenkins Pipelines, which can be scripted or defined using a domain-specific language (DSL).

11.1.4. Real-time Monitoring: Jenkins provides real-time feedback and logs for builds and tests.

11.1.2. Easy Installation and Configuration: It can be installed on various platforms and has a web-based user interface for configuration.

11.2. Common Use Cases:

Automating the build process of applications.

Running tests on code changes automatically.

Deploying applications to production or staging environments.

Jenkins is widely adopted in the software development community for its flexibility and robust capabilities in CI/CD workflows.

We can download the Jenkins from the jenkin's official website "Jenkins.io" for the specified OS used in the ec2 instance. In this project we have used Ubuntu server hence I have used docker for ubuntu server.

The steps and commands are as follows:

This is the Debian package repository of Jenkins to automate installation and upgrade. To use this repository, first add the key to your system:

sudo wget -O /usr/share/keyrings/Jenkins keyring.as\https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

Then add a Jenkins apt repository entry:

echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \ https://pkg.jenkins.io/debian-stable binary/ | sudo tee \ /etc/apt/sources.list.d/jenkins.list > /dev/null

Update your local package index, then finally install Jenkins:

sudo apt-get update

sudo apt-get install fontconfig openjdk-17-jre

sudo apt-get install Jenkins

sudo systemctl enable Jenkins

sudo systemctl start Jenkins

11.3. Unlocking the Jenkins:

Go to the http://publicipoftheserver:8080, we will get the below screen,

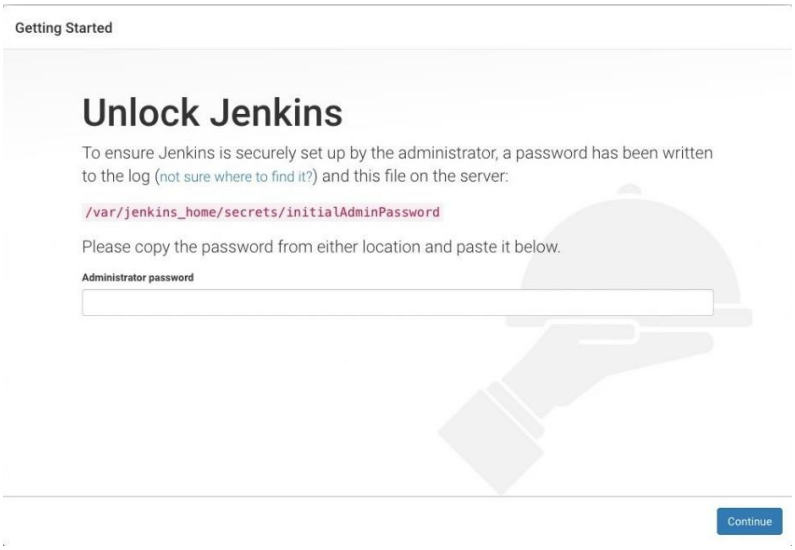

Fig 10.1. Unlock screen of Jenkins

Now cat to the path give in the screen in the Jenkins server and copy and paste the password in that path and give continue

Install the recommended plugins from the Jenkins

Now create a new user using a username, password,email ID and give start using Jenkins.

Now go to new Item and create a new pipeline job called feedback and the steps are as follows:

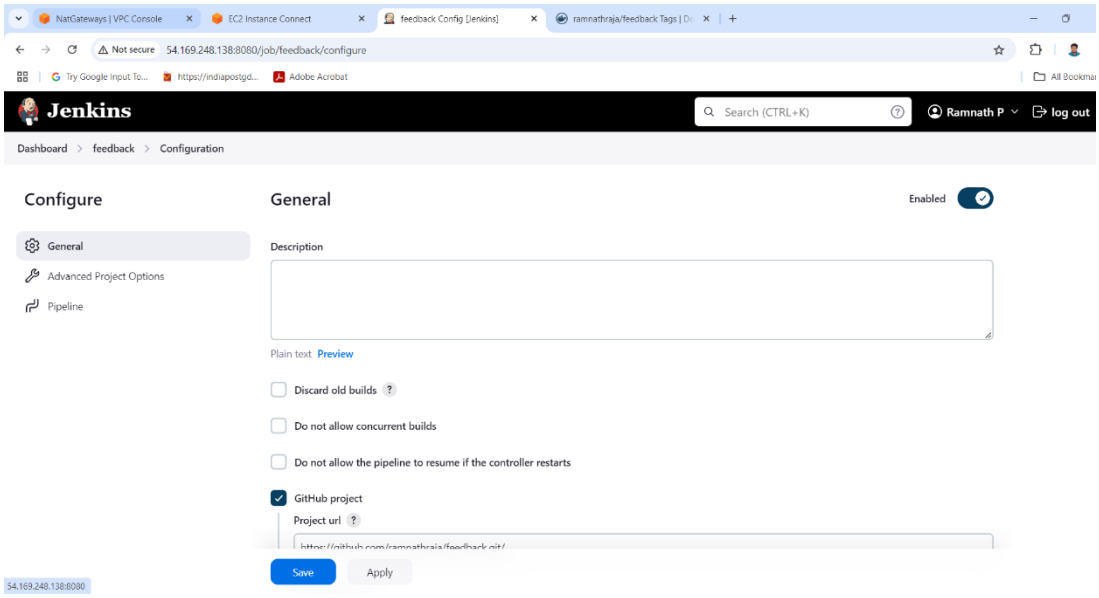

Fig 10.2. Configuration of Jenkins 1

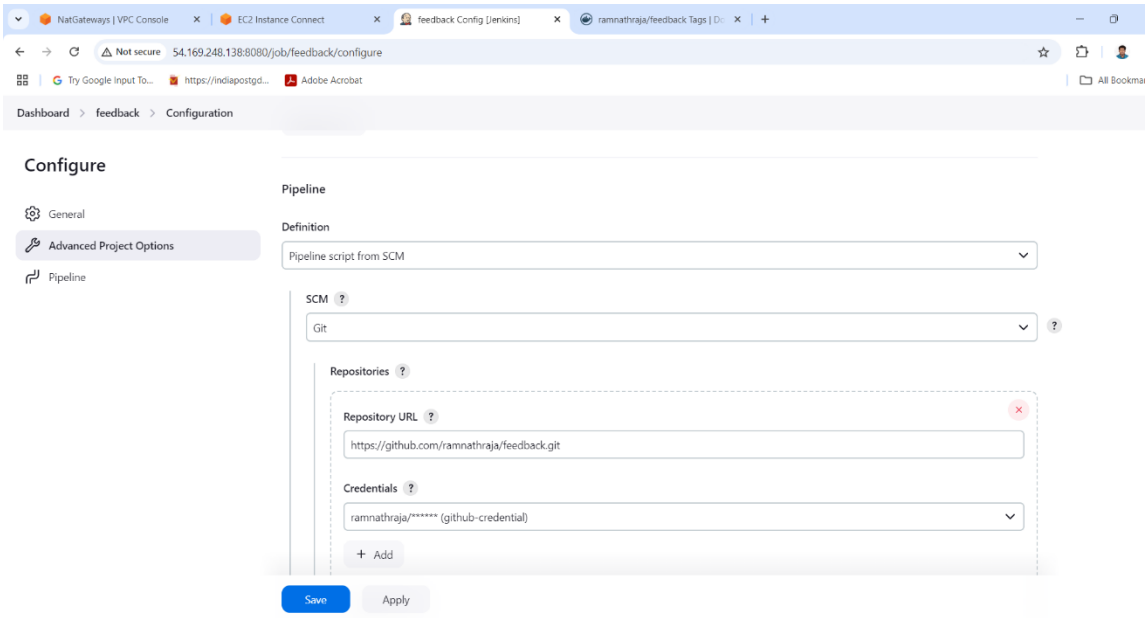

Fig 10.3. Configuration of Jenkins 2

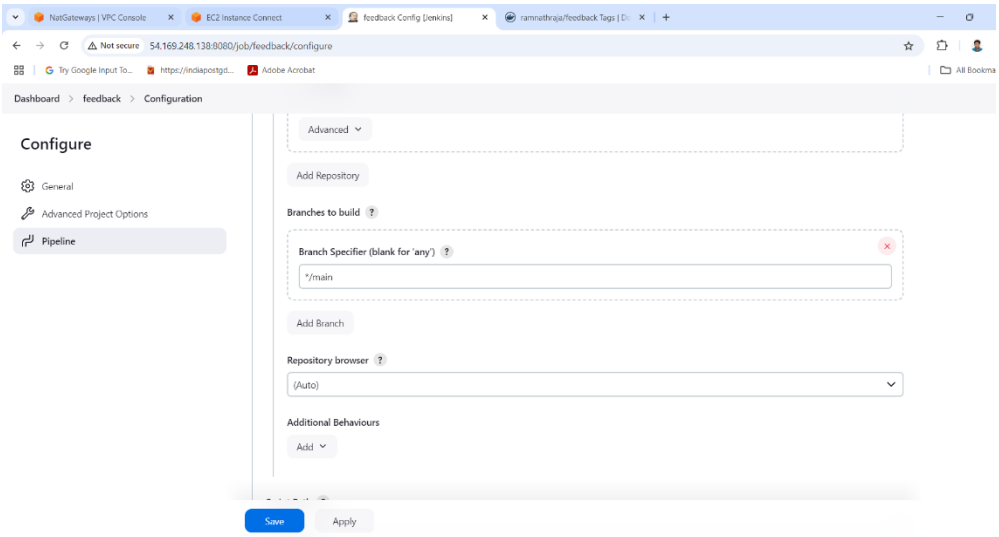

Fig 10.4. Configuration of Jenkins 3

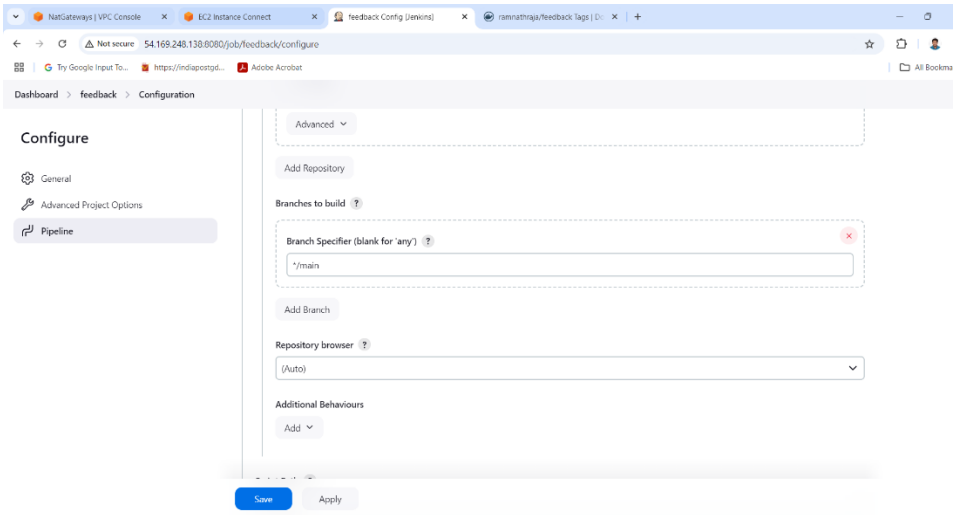

Fig 10.5. Configuration of Jenkins 4

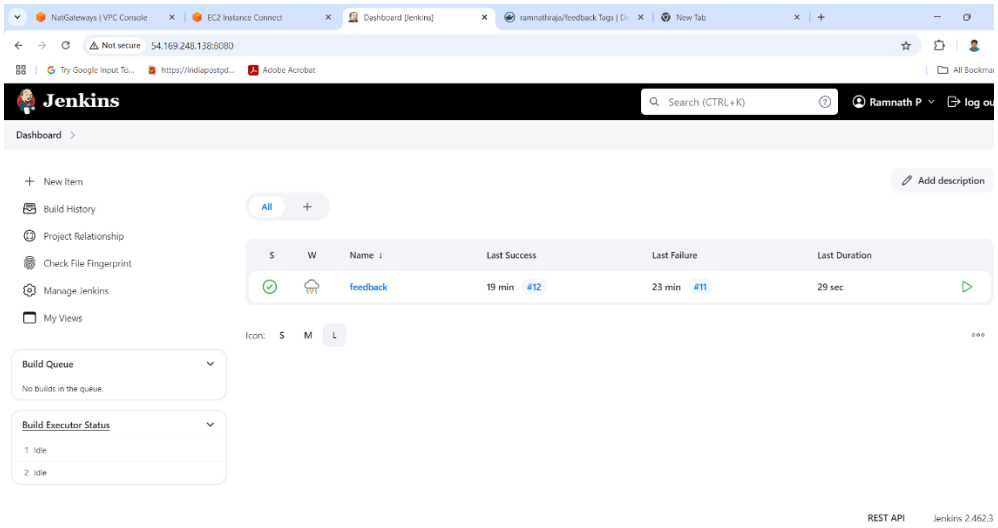

Fig 10.6. Configuration of Jenkins 5

Now for this pipeline we have to add the jenkinsfile to the github repository.

The Jenkinsfile script is as follows:

```
pipeline {
    agent any
    environment {
        KUBECONFIG = '/var/lib/jenkins/kubeconfig'
        AWS_DEFAULT_REGION = 'ap-southeast-1' // Update with your desired region
        EKS_CLUSTER_NAME = 'your-eks-cluster' // Name of your EKS cluster
        // KUBE_CONFIG = credentials('kubeconfig') // Jenkins credential for kubeconfig file
        DOCKER_IMAGE = 'ramnathraja/feedback:latest' // Docker image from Docker Hub
        KUBE_NAMESPACE = 'default' // Namespace in EKS cluster
        DEPLOYMENT_NAME = 'demo-deployment' // Name of the deployment
```

```
                registry = "ramnathraja/feedback"
                registryCredential = 'docker-hub-credentials'
                dockerImage = ''
    }

                stages {
                    stage('Git Checkout') {
                        steps {
                            script {
                                // Checkout the repository
                                git url: 'https://github.com/ramnathraja/feedback.git', branch: 'main'
                            }
                        }
                    }
                stage('Building Docker Image') {
                        steps {
                            script {
                                dockerImage = docker.build(registry + ":$BUILD_NUMBER")
                            }
                        }
                    }
                stage('Push Image To Docker Hub') {
                            steps {
                            script {
```

```
docker.withRegistry('https://registry.hub.docker.com', registryCredential) {
                    dockerImage.push("${env.BUILD_NUMBER}")
                                                                dockerImage.push("latest")
                }
            }
        }
    }
                                stage('Deploy to Kubernetes') {
                                                steps {
                                                withCredentials([[$class:
'AmazonWebServicesCredentialsBinding', credentialsId: 'aws-eks-credential']]) {
                                                                script {
                                                                        sh 'aws eks update-
kubeconfig --name my-eks-cluster --region ap-southeast-1'
                                                                                sh 'kubectl apply -f
deployment.yml'
                }
            }
        }
    }
  }
```

```
        post {
            always {
                cleanWs() // Clean up workspace after the build
        }
    }
}
```

11.4. Global Tool Configuration:

In Jenkins, you can set up tools globally to ensure that they are available for all jobs and pipelines. To configure tools globally:

Navigate to **Manage Jenkins**.

Select **Global Tool Configuration**.

Here, you can define different versions of:

JDK: Java Development Kit.

Git: The version control system.

Maven: The build automation tool for Java projects.

Gradle: Another build automation tool used primarily for Java.

NodeJS: JavaScript runtime.

Once tools are configured globally, they can be referenced in jobs without having to specify the installation paths.

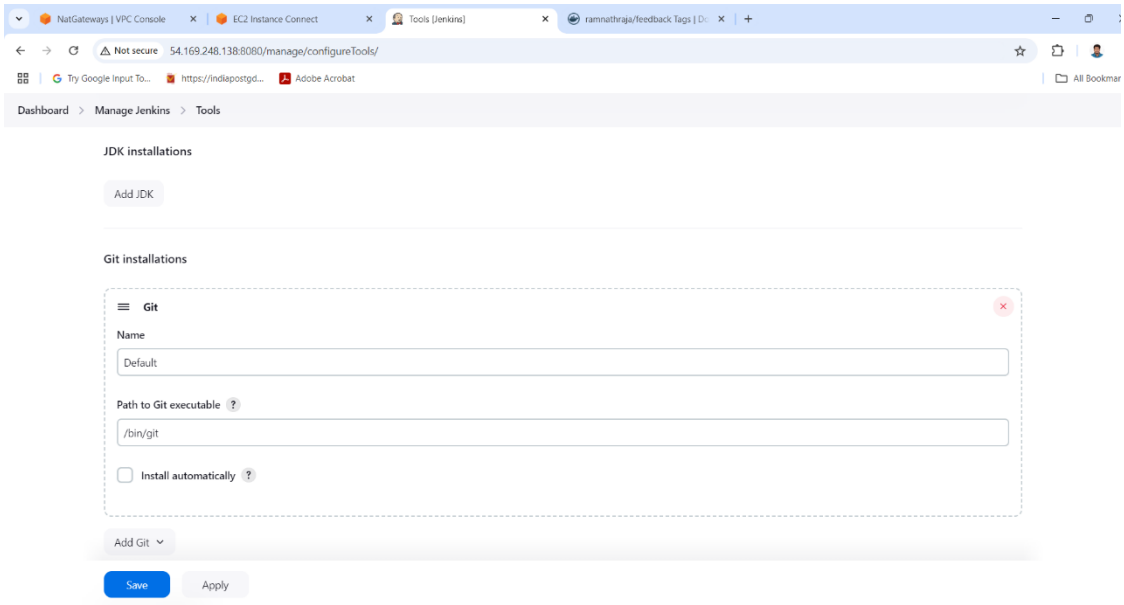

Fig 10.7. Configuration of Gobal Tool Configuration

11.5. Credentials for Jenkins:

In Jenkins, credentials are used to store sensitive information securely, such as usernames, passwords, SSH keys, or API tokens.

11.5.1. Steps to Add Credentials:

Go to Jenkins Dashboard → Manage Jenkins.

Select **Manage Credentials**.

Choose a **domain** (e.g., Global).

Click **Add Credentials**.

Select the type (e.g., Username and Password, Secret Text, SSH Key).

Fill in the required details (e.g., username, password, or secret).

Click **OK** to save.

You can reference these credentials in Jenkins jobs or pipelines securely.

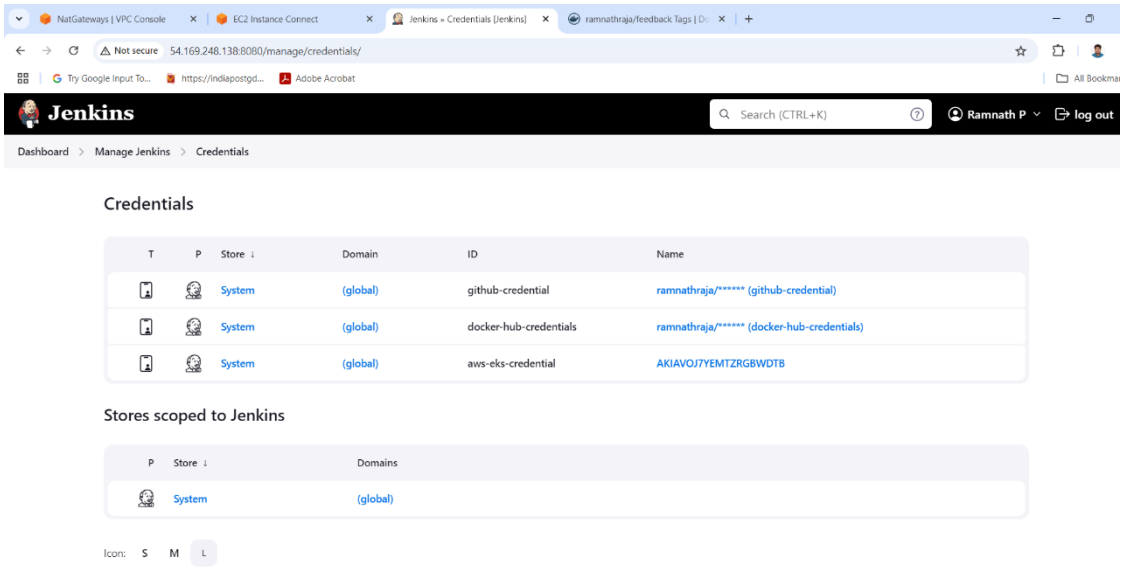

Fig 10.8. Credentials of the Jenkins

11.6. Nodes:

In Jenkins, **nodes** (also known as agents) are machines that execute build jobs. The main Jenkins server, called the **master** or **controller**, coordinates job scheduling and delegating tasks to nodes.

11.6.1. Types of Nodes:

11.6.1.1. Master (Controller): The primary Jenkins server that handles scheduling, monitoring nodes, and dispatching builds.

11.6.1.2. Agent (Node): Secondary machines that run build tasks assigned by the master.

11.6.2. Steps to Add a Node:

Go to Jenkins Dashboard → Manage Jenkins.

Click on **Manage Nodes and Clouds**.

Click **New Node**.

Define the node details (name, type as a permanent agent).

Set the remote root directory (where Jenkins will store build data on the node).

Define the Launch Method (SSH, command execution, etc.).

Save the configuration.

Nodes allow Jenkins to distribute build workloads across multiple machines, improving performance and scalability.

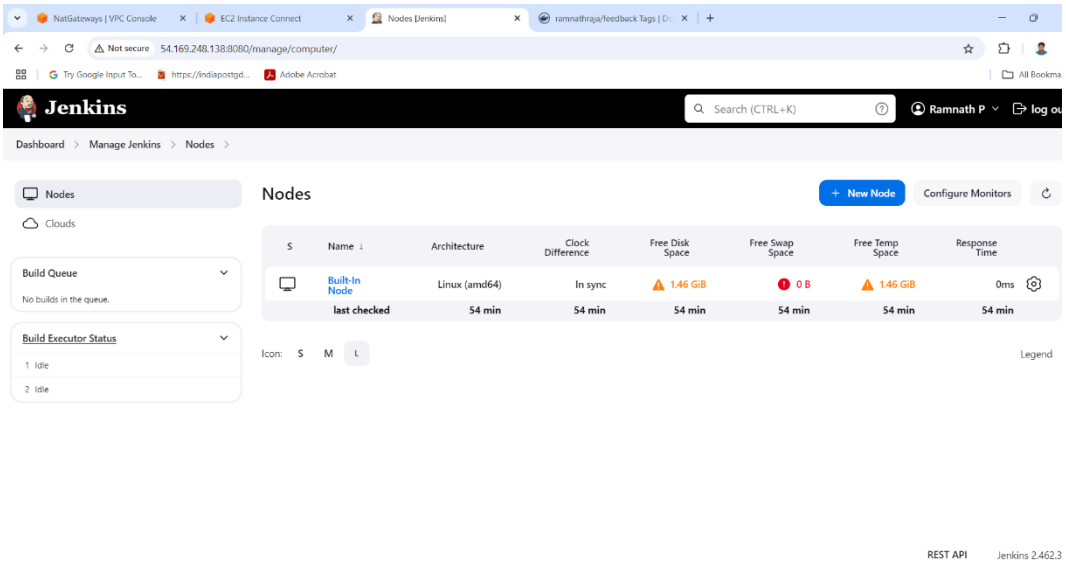

Fig 10.9. Node in Jenkins

Starting of the Jenkins Pipeline:

Now go to the feedback pipeline job and click the “build now”

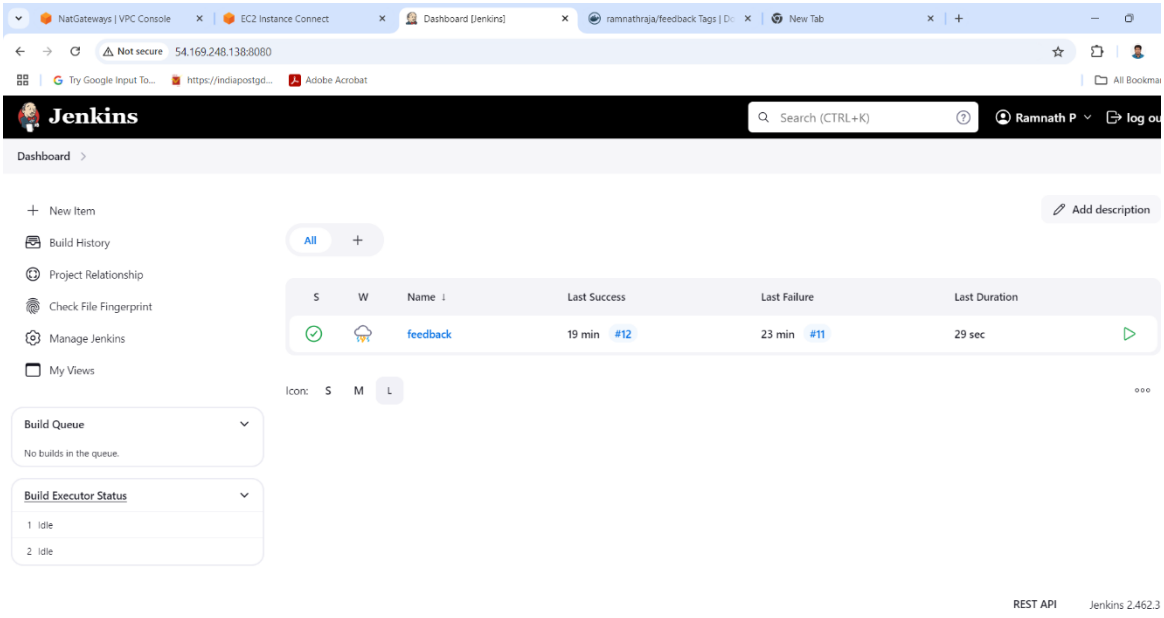

Fig 10.10. Feedback pipeline job

We can see the Success build in the Build Queue

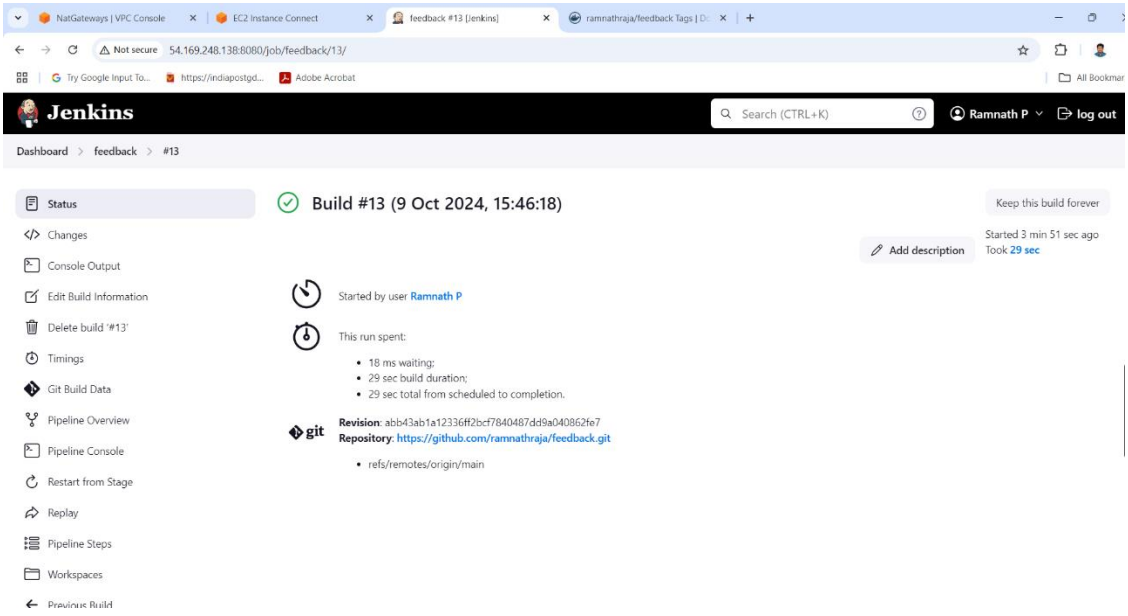

Fig 10.11. Successful Built

We can see the output of the build in the console output tab,

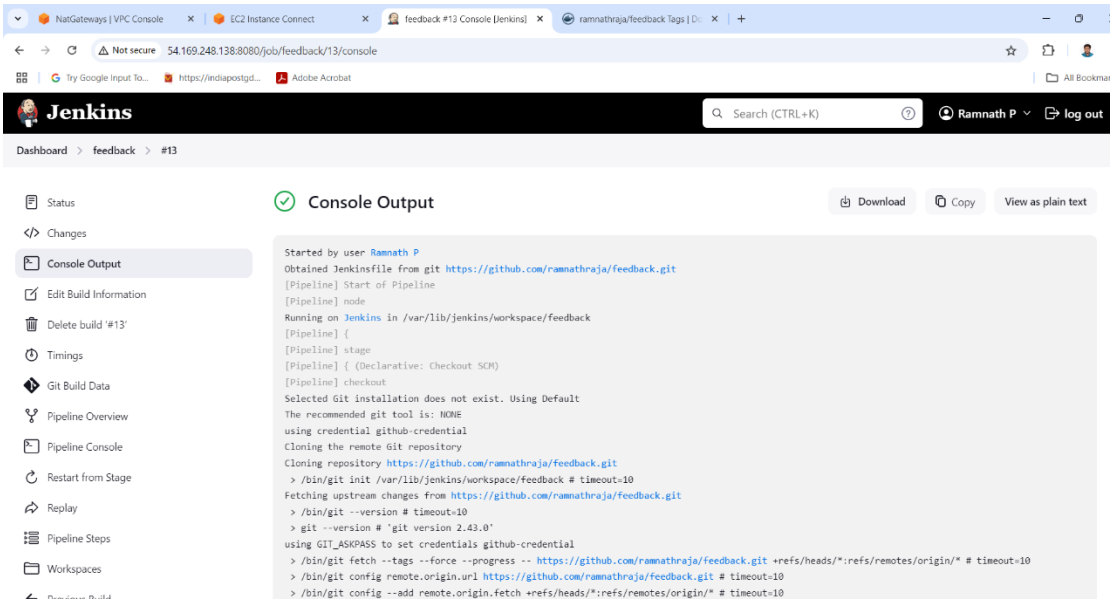

Fig 10.12. Successful Built console output 1

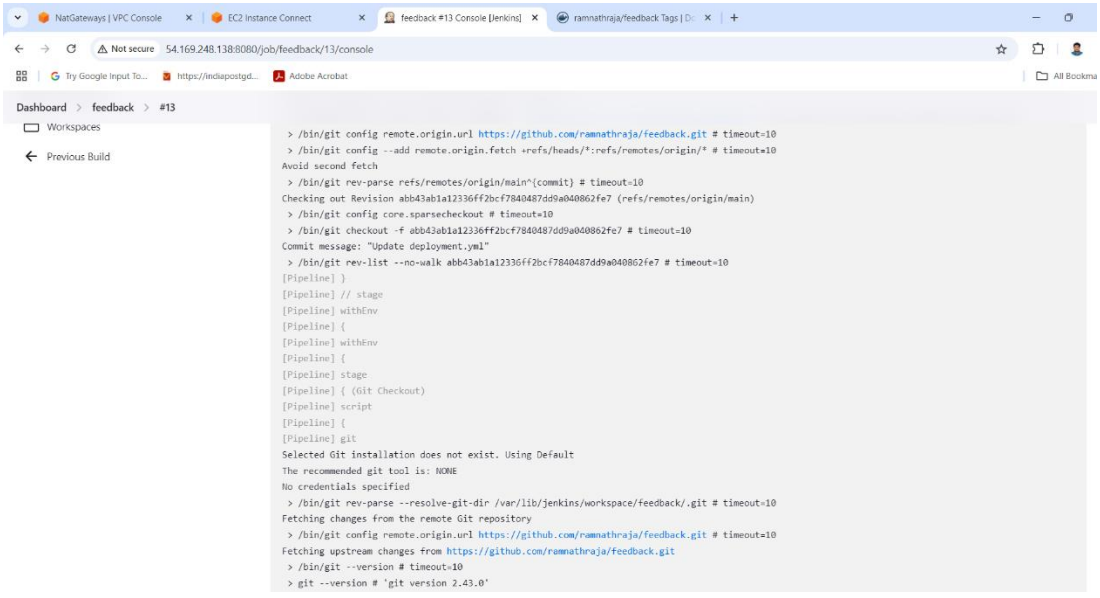

Fig 10.13. Successful Built console output 2

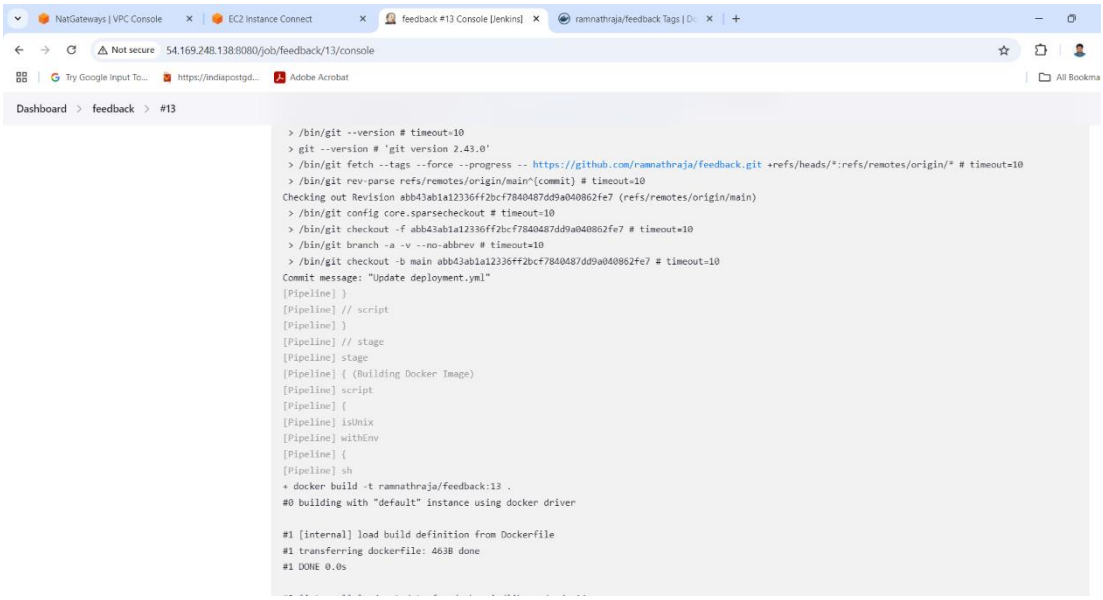

Fig 10.14. Successful Built console output 3

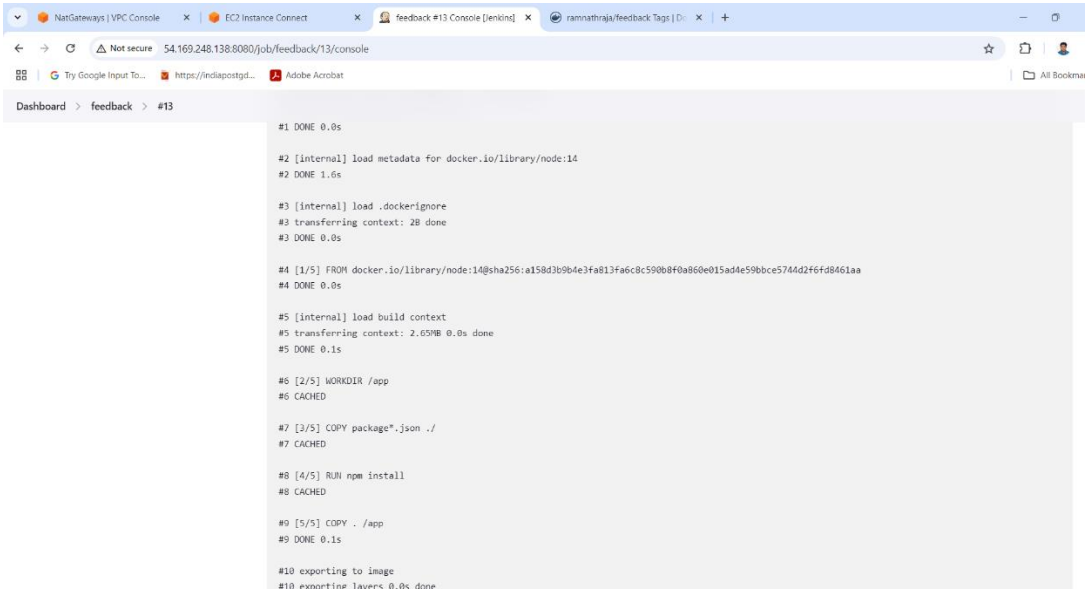

Fig 10.15. Successful Built console output 4

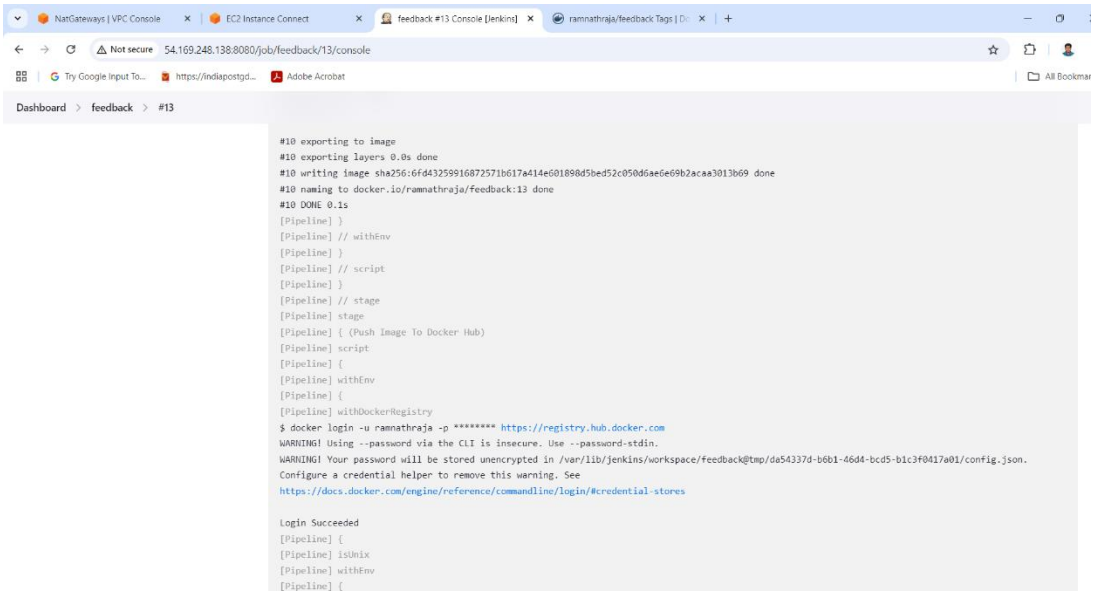

Fig 10.16. Successful Built console output 5

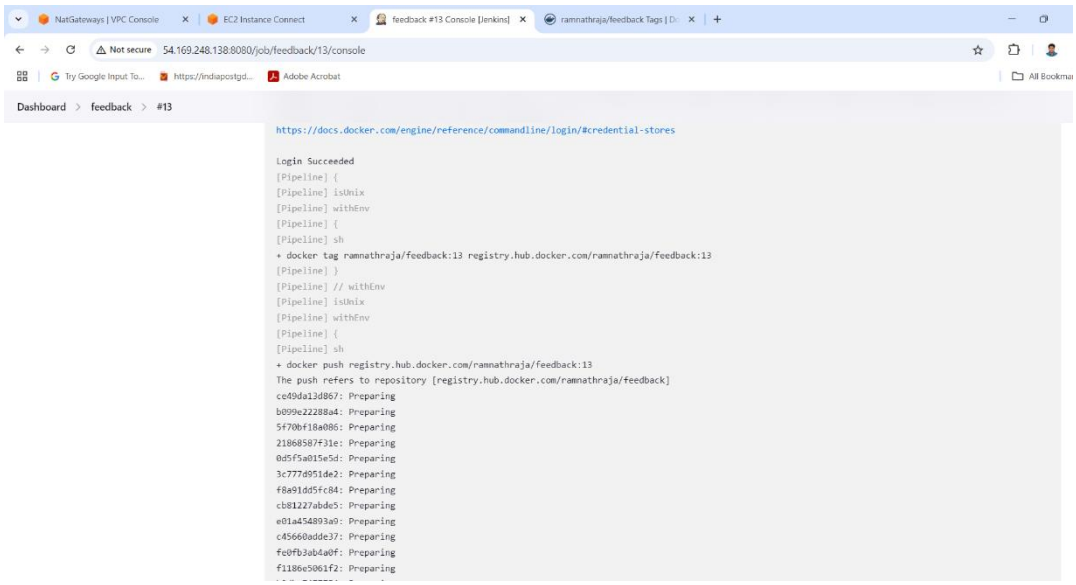

Fig 10.17. Successful Built console output 6

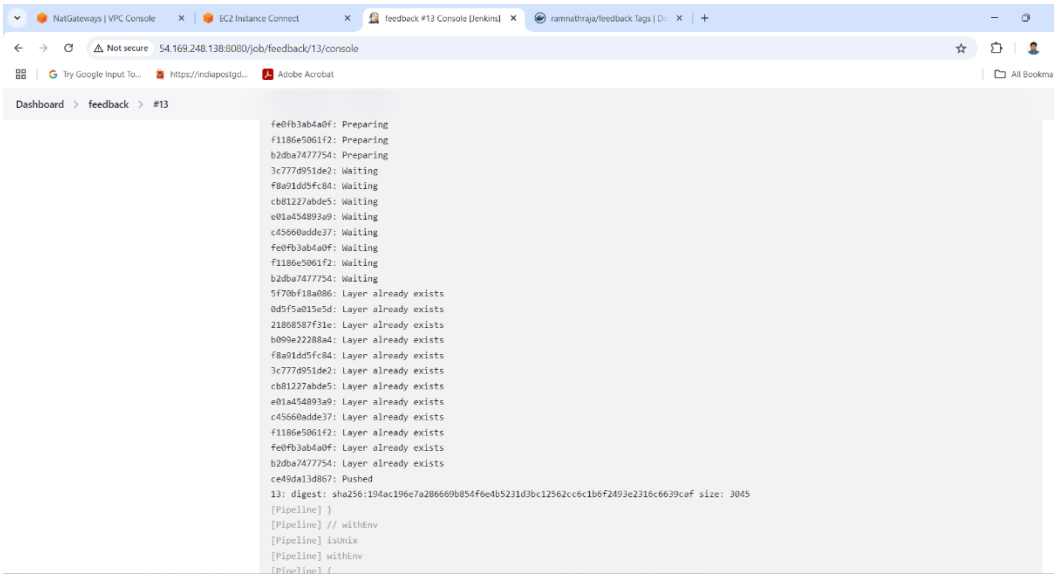

Fig 10.18. Successful Built console output 7

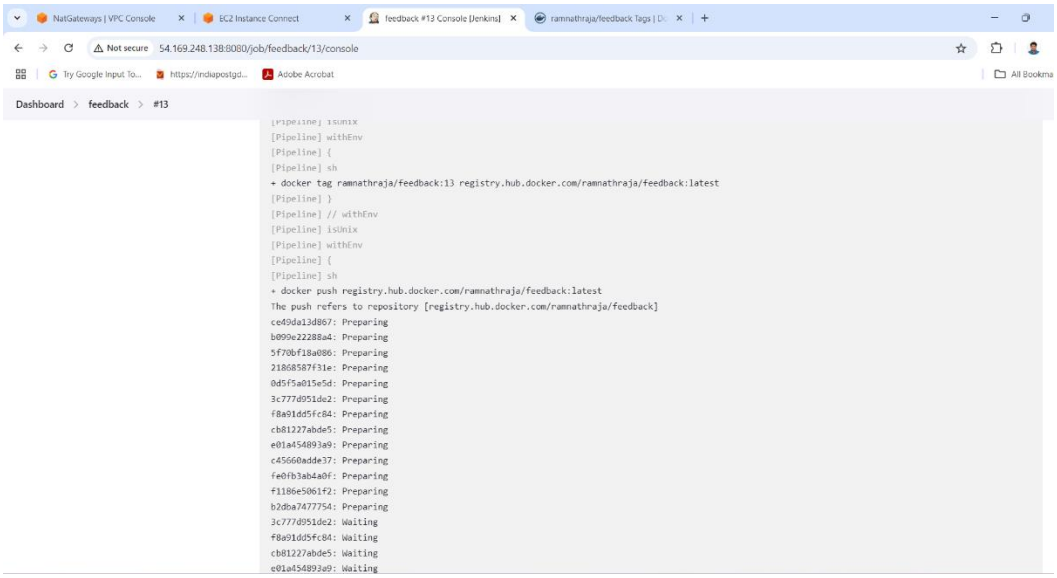

Fig 10.19. Successful Built console output 8

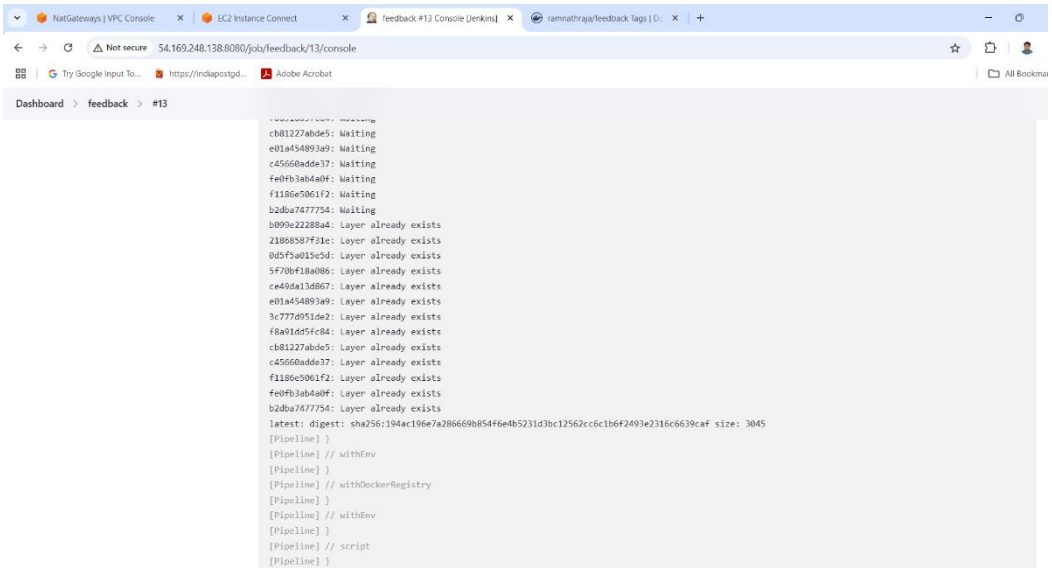

Fig 10.20. Successful Built console output 9

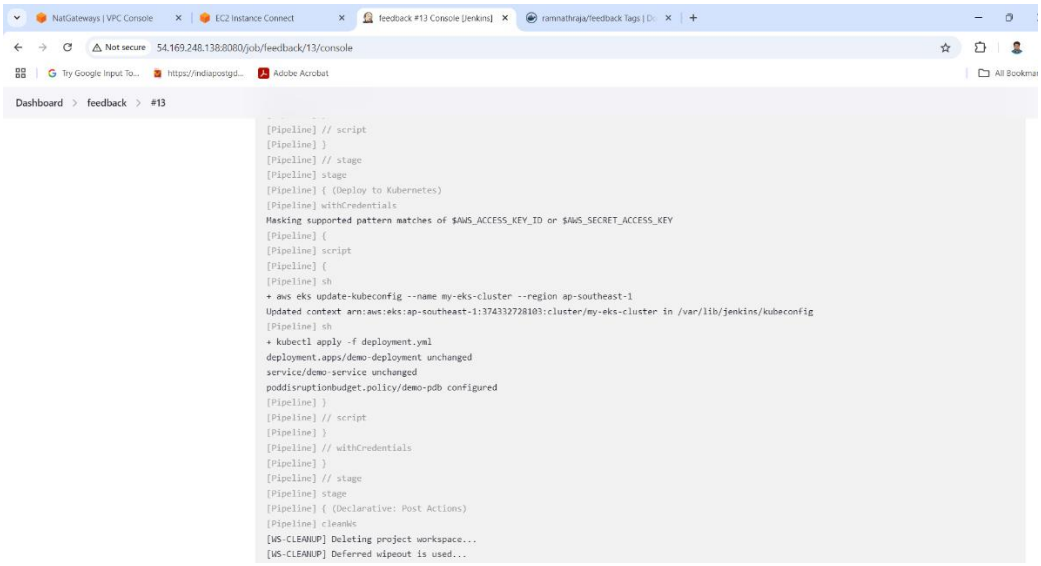

Fig 10.21. Successful Built console output 10

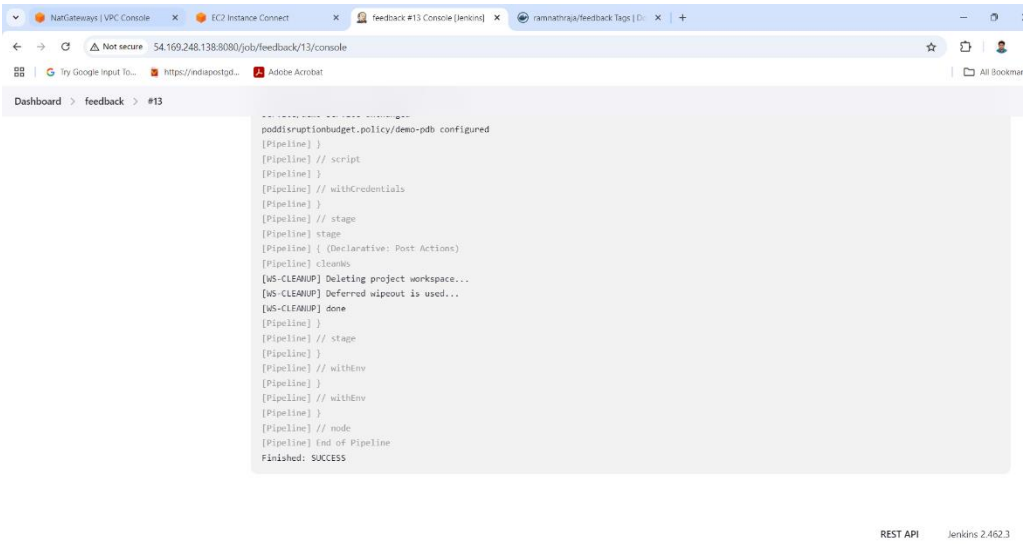

Fig 10.22. Successful Built console output 11

11.7. Issue and Work-Around:

The issue we may face is getting the default git installation tools is not found for this we need to assign the /bin/git path in the git section in the global tools configuration.

The other issue we may face is Authentication required issue for the Kubernetes, this may be rectified by either copying the admin.conf file in /etc/Kubernetes to the /var/lib/Jenkins path as kubeconfig file.And add the permission for the file so that the jenkins can access that file by giving the following commands,

```
chown jenkins:jenkins /var/lib/jenkins/kubeconfig
chmod 600 /var/lib/jenkins/kubeconfig
```

We can also face permission denied issue, this can be rectified by giving necessary permission for the Jenkins user.Put the following commands,

```
sudo usermod -aG docker jenkins

sudo systemctl restart jenkins

sudo systemctl status docker

ls -l /var/run/docker.sock
```

The output should be as follows,

srw-rw---- 1 root docker 0 Oct 10 12:34 /var/run/docker.sock

12. Docker Hub:

Docker Hub is a cloud-based repository service where Docker users can store, share, and manage Docker images. It serves as a central hub for finding and accessing pre-built images, such as application containers, libraries, and other useful components. Some key features include:

12.1. Public and Private Repositories: Users can create public repositories to share images with the community or private repositories for personal use or internal company projects.

12.2. Official Images: Docker Hub offers official images from verified publishers like Ubuntu, Nginx, MySQL, etc., which are maintained and trusted by Docker.

12.3. Automated Builds: Users can automatically build and push Docker images to Docker Hub from source code hosted in repositories like GitHub.

12.4. Search and Explore: Docker Hub provides a search interface to easily find existing images, saving time in creating your own from scratch.

12.5. Image Versioning: Users can manage different versions (tags) of images, making it easy to update or roll back to previous versions.

The image pushed from the Jenkins pipeline is pushed in the docker hub in the registry “ramnathraja/feedback”. For every built made in the Jenkins pipeline a latest image will be updated in the docker hub registry. The old images will be named with a tagname and it also stored in the same registry.

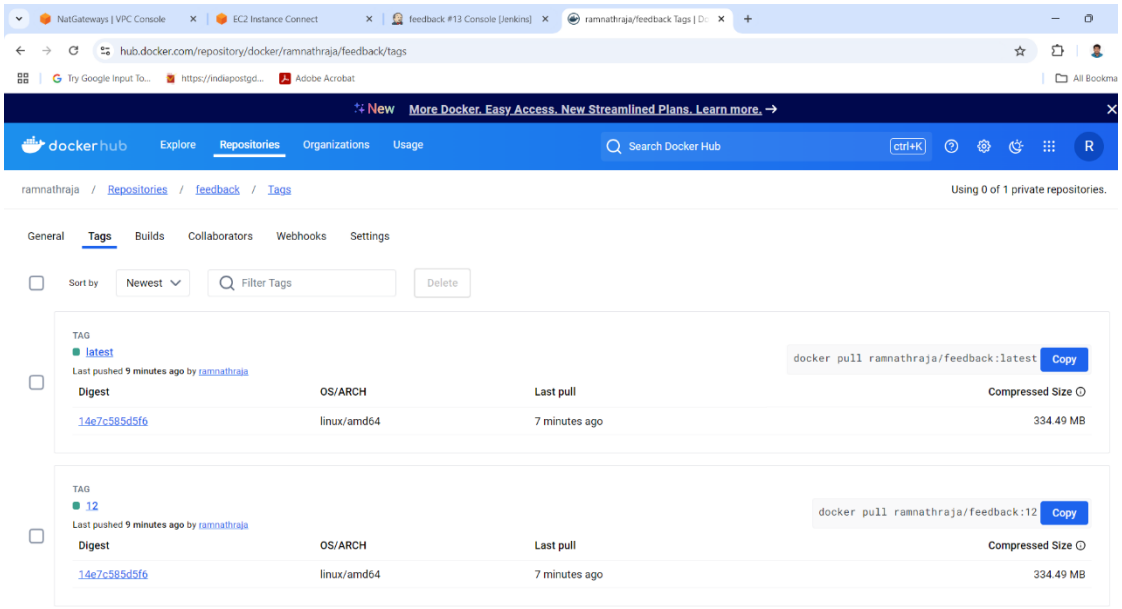

Fig 11. Docker Hub Registry

13. EKS Cluster:

Amazon EKS (Elastic Kubernetes Service) is a managed Kubernetes service that simplifies the process of running Kubernetes on AWS. EKS allows users to deploy, manage, and scale containerized applications using Kubernetes without needing to install, operate, or maintain their own Kubernetes control plane. Here's a brief overview:

13.1. Managed Kubernetes: EKS automates the setup of the Kubernetes control plane, including security, availability, and scalability, so you don't have to manage it manually.

13.2. Integration with AWS Services: EKS is tightly integrated with other AWS services, such as IAM for security, Elastic Load Balancing, VPC for networking, and CloudWatch for monitoring.

13.3. Scalability and Reliability: It offers high availability by distributing the control plane across multiple Availability Zones, and users can scale their applications with Kubernetes' built-in autoscaling features.

13.4. Security: EKS automatically applies the latest security patches to the Kubernetes control plane and integrates with AWS security services like IAM and Secrets Manager.

13.5. Flexible Workloads: EKS supports a wide variety of workloads, including microservices, batch processing, machine learning, and web applications.

EKS helps businesses focus on developing and deploying their containerized applications rather than managing the Kubernetes infrastructure.

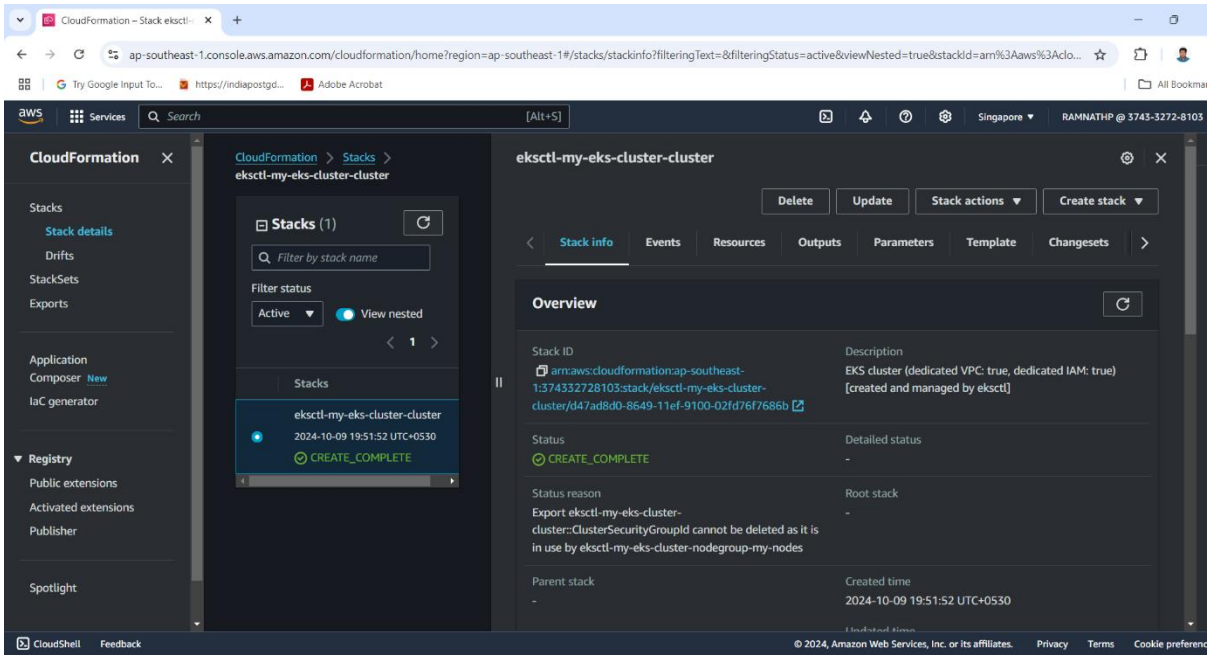

Fig 12. My-eks-cluster

14. Cloud Formation:

AWS CloudFormation is a service that allows you to model, provision, and manage AWS resources using templates written in either JSON or YAML. It simplifies infrastructure management by enabling Infrastructure as Code (IaC), allowing you to automate the setup of AWS environments. Here's a brief overview:

14.1. Infrastructure as Code: CloudFormation enables you to define AWS resources and infrastructure (such as EC2 instances, S3 buckets, VPCs, etc.) in a declarative way using templates. This means you can version-control, share, and reuse infrastructure definitions.

14.2. Automated Provisioning: Once a template is created, CloudFormation can automatically create, update, or delete resources across AWS services, following the configurations specified in the template.

14.3. Stack Management: Resources are organized into "stacks," which are collections of related AWS resources. You can easily manage, update, or delete all resources in a stack together.

14.4. Repeatability and Consistency: Using CloudFormation ensures consistent setups for environments (e.g., dev, test, prod) by deploying the same templates multiple times with predictable results.

14.5. Rollback and Changesets: When updating a stack, you can preview changes and roll back in case of failures, helping to minimize downtime and errors.

CloudFormation is ideal for automating infrastructure deployment, ensuring repeatable, error-free environment setups across teams.

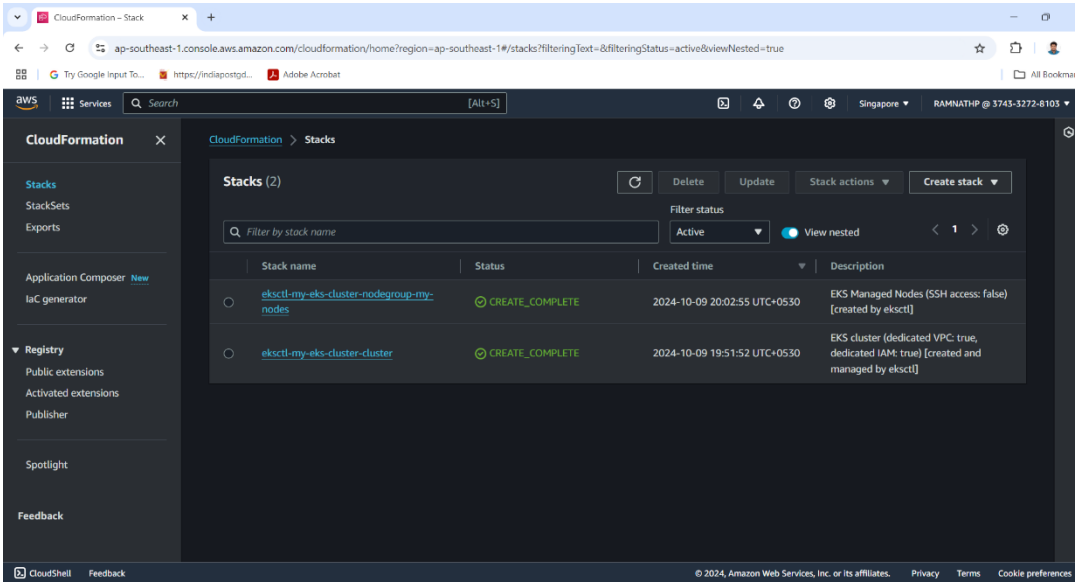

Fig 13. Cloud Formation Stack

15. Kubernetes Deployment:

A **Kubernetes Deployment** is a resource in Kubernetes that manages the lifecycle of applications, ensuring that the desired state of an application is maintained consistently. It is used to declare how many replicas of an application should run, how to update or roll back the application, and more. Here's an overview:

15.1. Purpose: A deployment ensures that a specified number of replicas (instances) of an application are running and replaces failed or unhealthy pods automatically.

15.2. Declarative Management: You declare the desired state of your application in a YAML or JSON configuration file, and Kubernetes ensures that the actual state matches it.

15.3. Scaling: Deployments make it easy to scale applications by adjusting the number of replicas dynamically. You can increase or decrease the number of pods (replicas) to match the traffic or load.

15.4. Rolling Updates: A deployment allows for seamless updates to applications with minimal downtime through **rolling updates**. This ensures new versions of an application are deployed gradually, replacing old versions without service disruption.

15.5.Rollback: If an update fails or if there are issues with a new version of the application, Kubernetes Deployments provide a simple way to **roll back** to a previous stable version.

15.6.Self-Healing: If any pod in the deployment crashes or becomes unresponsive, Kubernetes automatically replaces it to ensure the desired number of replicas is always running.

16. Deployment File:

A **Kubernetes Deployment file** is typically a YAML (or JSON) configuration file that defines how an application should be deployed, including the number of replicas, the container image, resource limits, labels, and other important configurations. The Deployment file used in this project is named as deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-deployment
```

```
  labels:
    app: demo-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo-app
  template:
    metadata:
      labels:
        app: demo-app
    spec:
      containers:
      - name: demo-app
        image: ramnathraja/feedback
        ports:
        - containerPort: 8081
---
apiVersion: v1
kind: Service
metadata:
  name: demo-service
spec:
  selector:
    app: demo-app
  ports:
```

```
    - port: 8081
      targetPort: 8081
      nodePort: 30008
  type: LoadBalancer
---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: demo-pdb
spec:
  minAvailable: 1
  selector:
    matchLabels:
      app: demo-app
```

After the successful deployment of this file through Jenkins pipeline we can find the deployment in the Kubernetes server ie. Jenkins server in this case. Through this deployment we can access the application in the browser.

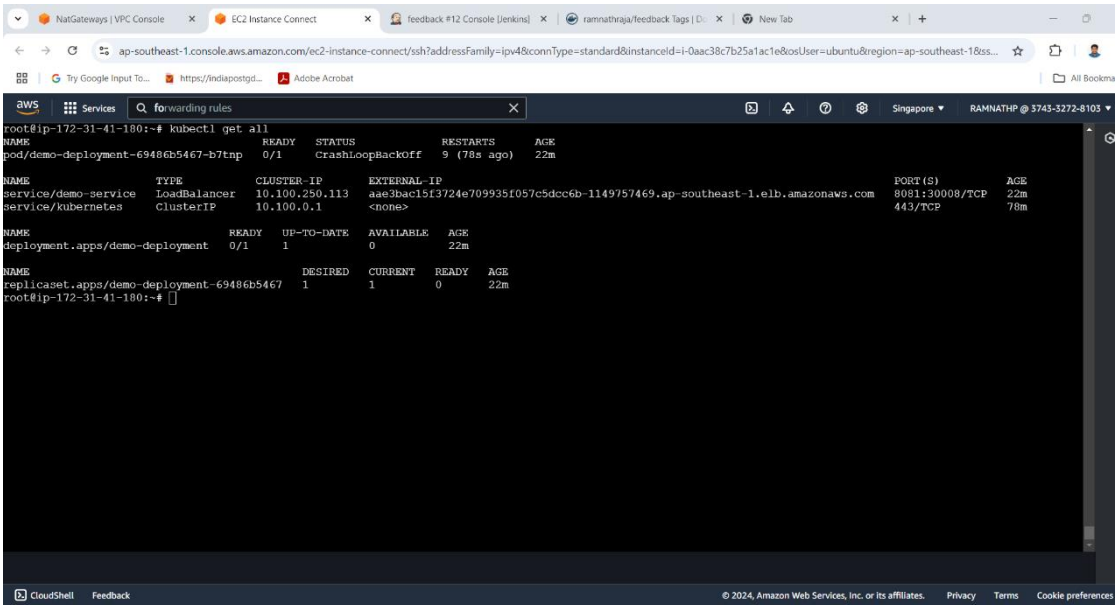

Fig 14. Deployment Output

16.1. Note:

This project helps us to convert a github project into a successful deployment in the Kubernetes server and access the application.

We can also install Prometheus (Monitoring Tool) and Grafana (Visualization Tool) for monitoring the server meterics of the Jenkins and Kubernetes tools.