

# KUBERNETES DEPLOYMENT PIPELINE FOR FEEDBACK APP WITH JENKINS AND KOPS

Github to Kubernetes

Ramnath P  
Ramnathraja935@gmail.com  
+91 9791438315

## Table Of Content

S.No	Topics	Page No
1	Abstract	6
2	Virtual Private Cloud	7
3	Subnet	7
4	Route table	8
5	Internet Gateway	9
6	NAT Gateway	10
7	Elastic IP	11
	7.1 Static Nature	11
	7.2 Association	11
	7.3 Reassignment	11
	7.4 Cost	11
	7.5 Use Cases	11
8	Github and Git	12
	8.1 Git	12
	8.2 Key Features	12
	8.3 Example Commands	13
	8.4 GitHub	13
	8.5 Key Features	13
	8.5.1 Remote Repositories	13
	8.5.2 Pull Requests	13
	8.5.3 Forking	13
	8.5.4 CI/CD Integration	13
9	Installing Docker	14
	9.1 Dockerfile	16

10	Installing the kubeadm	17
10.1	The installation steps	17
10.2	Installing kubeadm, kubelet and kubectl	18
10.3	Creating a cluster with kubeadm	19
10.3.1	Installing a Pod network add-on	19
10.4	Download and install eksctl	20
10.5	Issue and Workaround	23
11	Installing Jenkins	24
11.1	Key Features	24
11.1.1	Extensibility	24
11.1.2	Distributed Builds	24
11.1.3	Pipeline Support	24
11.1.4	Real-time Monitoring	24
11.1.2	Easy Installation and Configuration	24
11.2	Common Use Cases	24
11.3	Unlocking the Jenkins	26
11.4	Global Tool Configuration	32
11.5	Credentials for Jenkins	33
11.5.1	Steps to Add Credentials	33
11.6	Nodes	34
11.6.1	Types of Nodes	34
11.6.1.1	Master (Controller)	34
11.6.1.2	Agent (Node)	34
11.6.2	Steps to Add a Node	34
11.7	Issue and Work-Around	42
12	Docker Hub	43

12.1	Public and Private Repositories	43
12.2	Official Images	43
12.3	Automated Builds	43
12.4	Search and Explore	43
12.5	Image Versioning	43
13	EKS Cluster	44
13.1	Managed Kubernetes	44
13.2	Integration with AWS Services	44
13.3	Scalability and Reliability	44
13.4	Security	44
13.5	Flexible Workloads	45
14	Cloud Formation	45
14.1	Infrastructure as Code	45
14.2	Automated Provisioning	45
14.3	Stack Management	46
14.4	Repeatability and Consistency	46
14.5	Rollback and Changesets	46
15	Kubernetes Deployment	46
15.1	Purpose	47
15.2	Declarative Management	47
15.3	Scaling	47
15.4	Rolling Updates	47
15.5	Rollback	47
15.6	Self-Healing	47
16	Deployment File	47
16.1	Note	50

## Table Of Content

S.No	Topics	Page No
Fig 1	VPC of the project	7
Fig 2	Subnet of the project	8
Fig 3	Route Table of the project	9
Fig 4	Internet Gateway of the project	10
Fig 5	NAT Gateway of the project	11
Fig 6	Elastic IP Address of the project	12
Fig 7	Github repo of the project	14
Fig 8	AWS configure command output	20
Fig 9.1	Creation of EKS Cluster 1	21
Fig 9.2	Creation of EKS Cluster 2	21
Fig 9.3	Creation of EKS Cluster 3	22
Fig 9.4	Creation of EKS Cluster(Correct output 1)	22
Fig 9.5	Creation of EKS Cluster(Correct output 2)	23
Fig 9.6	Creation of EKS Cluster(Correct output 3)	23
Fig 10.1	Unlock screen of Jenkins	26
Fig 10.2	Configuration of Jenkins 1	27
Fig 10.3	Configuration of Jenkins 2	27
Fig 10.4	Configuration of Jenkins 3	28
Fig 10.5	Configuration of Jenkins 4	28
Fig 10.6	Configuration of Jenkins 5	29
Fig 10.7	Configuration of Gobal Tool Configuration	33
Fig 10.8	Credentials of the Jenkins	34
Fig 10.9	Node in Jenkins	35
Fig 10.10	Feedback pipeline job	36

Fig 10.11	Successful Built	36
Fig 10.12	Successful Built console output 1	37
Fig 10.13	Successful Built console output 2	37
Fig 10.14	Successful Built console output 3	38
Fig 10.15	Successful Built console output 4	38
Fig 10.16	Successful Built console output 5	39
Fig 10.17	Successful Built console output 6	39
Fig 10.18	Successful Built console output 7	40
Fig 10.19	Successful Built console output 8	40
Fig 10.20	Successful Built console output 9	41
Fig 10.21	Successful Built console output 10	41
Fig 10.22	Successful Built console output 11	42
Fig 11	Docker Hub Registry	44
Fig 12	My-eks-cluster	45
Fig 13	Cloud Formation Stack	46
Fig 14	Deployment Output	50

## **1. Abstract:**

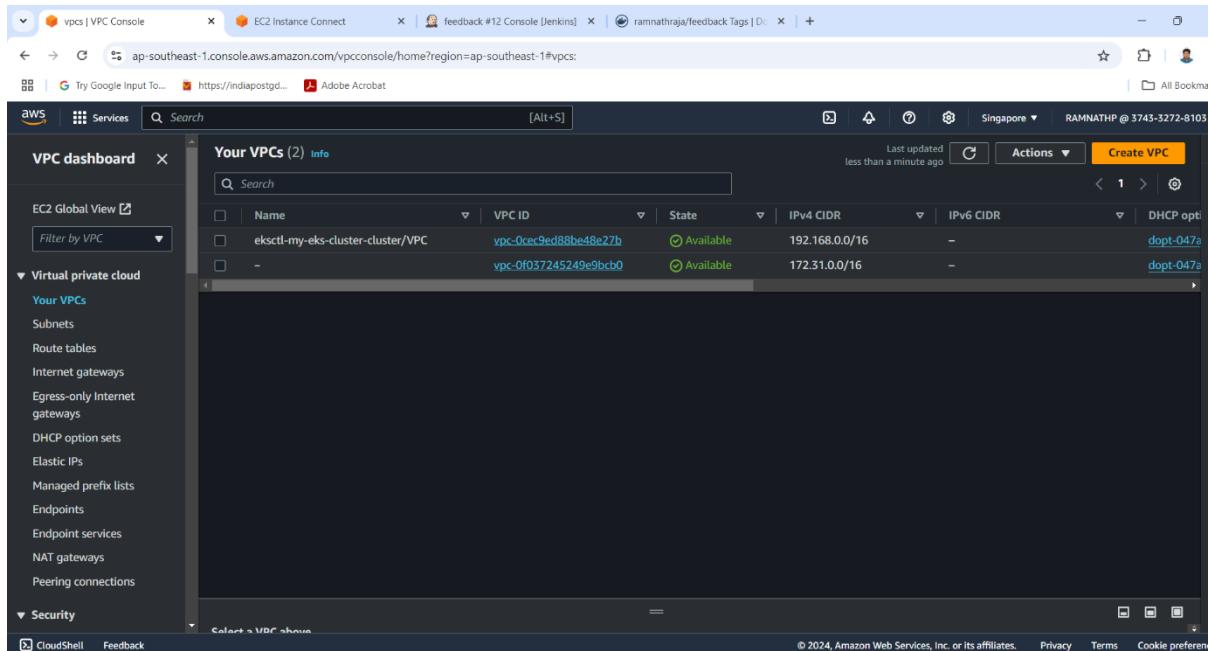
This project aims to streamline the deployment process of a GitHub project onto a Kubernetes server, facilitating seamless access to the application for users and developers alike. In today's fast-paced software development landscape, efficient deployment is crucial for maintaining competitive advantage 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.



The screenshot shows the AWS VPC Dashboard. The left sidebar lists various network components: EC2 Global View, Virtual private cloud (with sub-options like Your VPCs, Subnets, Route tables, Internet gateways, Egress-only Internet gateways, DHCP option sets, Elastic IPs, Managed prefix lists, Endpoints, Endpoint services, NAT gateways, and Peering connections), and Security. The main content area displays a table titled "Your VPCs (2) Info". The table has columns for Name, VPC ID, State, IPv4 CIDR, IPv6 CIDR, and DHCP opt. There are two entries: "eksctl-my-eks-cluster-cluster/VPC" with VPC ID "vpc-0ced88be48e27b" and state "Available", and another entry with VPC ID "vpc-0f037245249ebcb0" and state "Available". The IPv4 CIDR for the first VPC is "192.168.0.0/16" and for the second is "172.31.0.0/16". The IPv6 CIDR for both is "-". The DHCP options for the first VPC are "dopt-047a" and for the second are "dopt-047a".

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

Name	Subnet ID	State	VPC	IPv4 CIDR	IP5 CIDR
	subnet-001a33270ff74271	Available	vpc-0f037245249e9bcb0	172.31.0.0/20	
	subnet-0b8cf3fcab06d60cb	Available	vpc-0ec9ed88be48e27b   eks...	192.168.128.0/19	
	subnet-0fee730edb2dc7a8	Available	vpc-0ec9ed88be48e27b   eks...	192.168.0.0/19	
-	subnet-09f980c2cb4566747	Available	vpc-0f037245249e9bcb0	172.31.32.0/20	
	subnet-070d2f8420be92a5f	Available	vpc-0ec9ed88be48e27b   eks...	192.168.64.0/19	
	subnet-0257a1f67c15dd299	Available	vpc-0ec9ed88be48e27b   eks...	192.168.32.0/19	
	subnet-0643472992c3aed77	Available	vpc-0ec9ed88be48e27b   eks...	192.168.96.0/19	
	subnet-0f26b43fb8891be1	Available	vpc-0ec9ed88be48e27b   eks...	192.168.160.0/19	
-	subnet-0caeaf567986f3d4	Available	vpc-0f037245249e9bcb0	172.31.16.0/20	

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

The screenshot shows the AWS VPC Route Tables console. The left sidebar has a 'Virtual private cloud' section with 'Route tables' selected. The main area is titled 'Route tables (6) info' and lists six route tables:

Name	Route table ID	Explicit subnet association	Edge associations	Main	VPC
eksctl-my-eks-cluster-cluster/PrivateRo...	rtb-00faeb3d0c39329ab	subnet-0f26b43fb8891b...	-	No	vpc-0cec9ed88be48e27b   ek
-	rtb-04894c750fbe4450	-	-	Yes	vpc-0fd37245249e9bcbb
-	rtb-0ad3b08551faaa105	-	-	Yes	vpc-0cec9ed88be48e27b   ek
eksctl-my-eks-cluster-cluster/PrivateRo...	rtb-0b17a82ae4acbf4c	subnet-0643472992c5ae...	-	No	vpc-0cec9ed88be48e27b   ek
eksctl-my-eks-cluster-cluster/PublicRou...	rtb-05b0ea8122233dc	3 subnets	-	No	vpc-0cec9ed88be48e27b   ek
eksctl-my-eks-cluster-cluster/PrivateRo...	rtb-0b9047d388139b748	subnet-0b9cf3fcab06d60...	-	No	vpc-0cec9ed88be48e27b   ek

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.

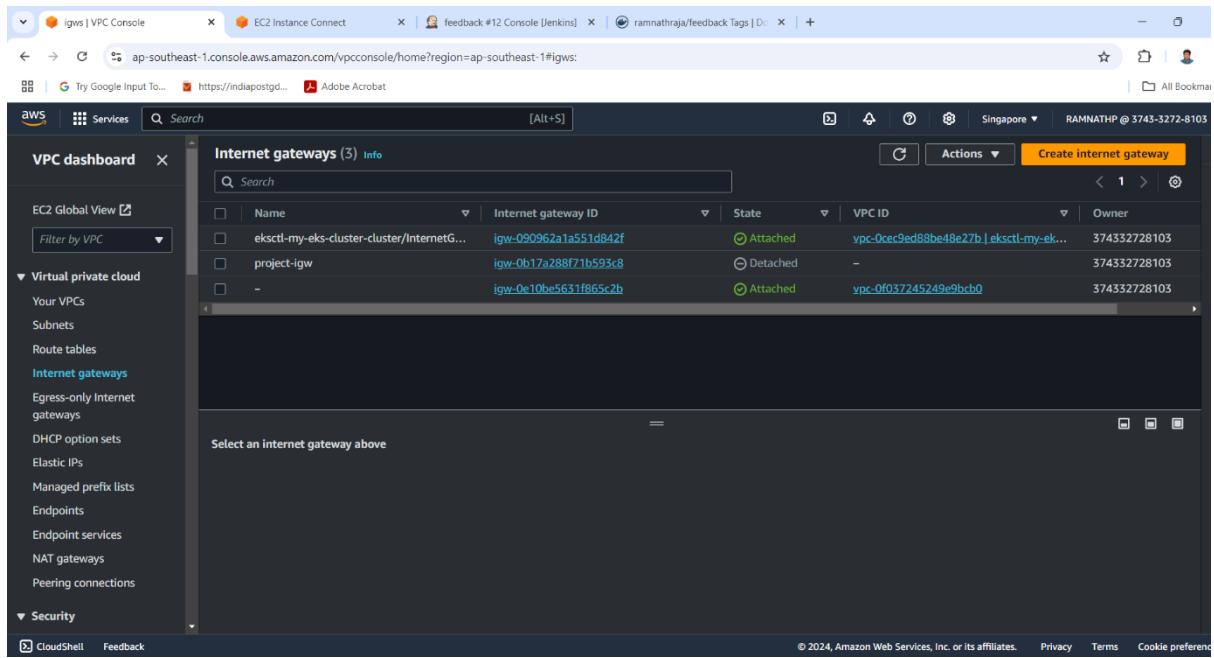


Fig 4. Internet Gateway of the project

## 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.

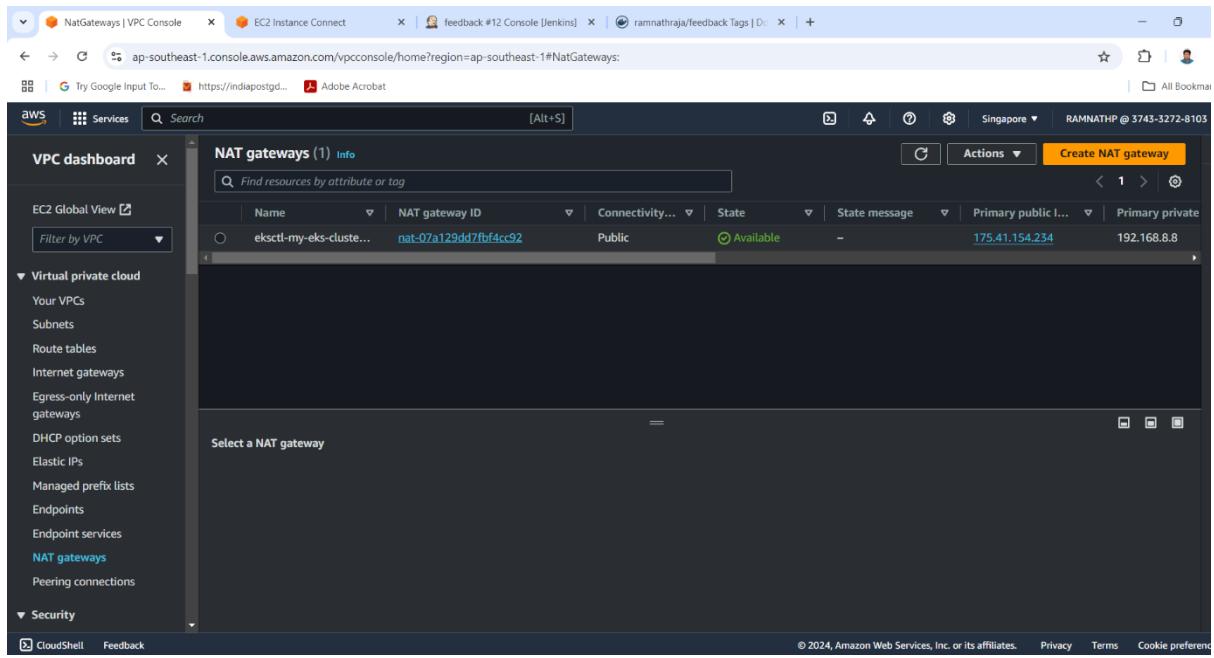


Fig 5. NAT Gateway of the project

## 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".

The screenshot shows the AWS VPC dashboard with the 'Elastic IP addresses' section selected. There is one entry in the table:

Name	Allocated IPv4 addr...	Type	Allocation ID	Reverse DNS record
eksctl-my-eks-cluster-cluster/NATIP	175.41.154.234	Public IP	eipalloc-0b1ea7faa3ceaa6ac	-

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

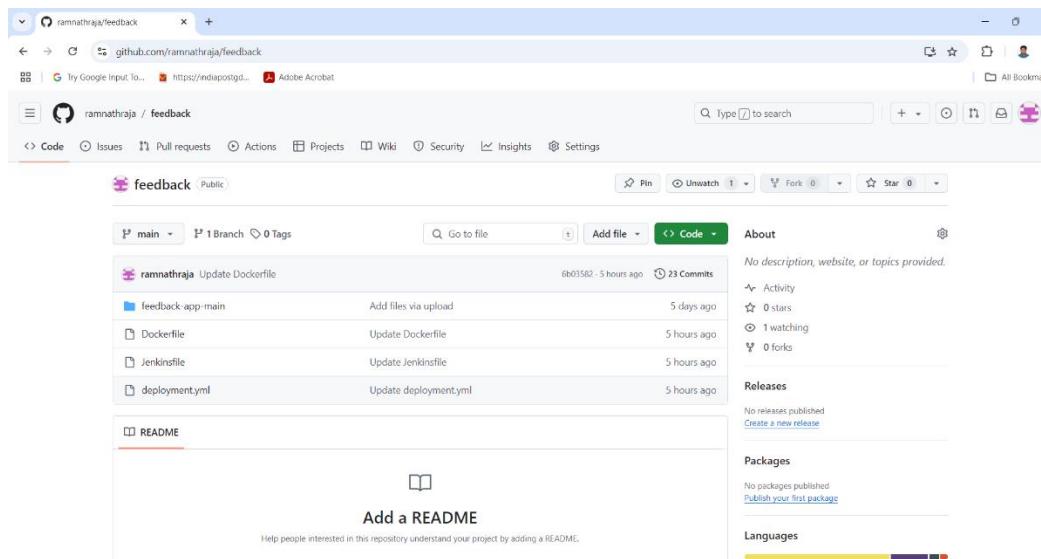


Fig 7. Github repo of the project

## 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](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](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](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 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

```
aws configure
AWS Access Key ID [*****WDTB*]:
AWS Secret Access Key [*****5XJ7*]:
Default region name [ap-southeast-1]: ap-southeast-1
Default output format [json]: json
```

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
```

```

inflating: aws/dist/docutil/writers/s5_html/themes/default/operate.css
root@ip-172-31-41-180:~# eksctl create cluster --name my-eks-cluster --region ap-southeast-1 --nodegroup-name my-nodes --nodes 2 --nodes-min 1 --nodes-max 2 --managed
eksctl: command not found
root@ip-172-31-41-180:~# curl -sSL "https://github.com/weaveworks/eksctl/releases/download/v0.147.0/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /usr/local/bin
root@ip-172-31-41-180:~# curl -sL "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_Linux_amd64.tar.gz" -o eksctl.tar.gz
root@ip-172-31-41-180:~# sudo mv eksctl /usr/local/bin
root@ip-172-31-41-180:~# eksctl version
0.191.0
root@ip-172-31-41-180:~# eksctl create cluster --name my-eks-cluster --region ap-southeast-1 --nodegroup-name my-nodes --nodes 2 --nodes-min 1 --nodes-max 2 --managed
eksctl version 0.191.0
using region ap-southeast-1
setting availability zones to [ap-southeast-1b ap-southeast-1a ap-southeast-1c]
subnets for ap-southeast-1b - public:192.168.0.0/19 private:192.168.96.0/19
subnets for ap-southeast-1a - public:192.168.32.0/19 private:192.168.128.0/19
subnets for ap-southeast-1c - public:192.168.64.0/19 private:192.168.160.0/19
nodegroup "my-nodes" will use "" [AmazonLinux2/1.30]
using Kubernetes version 1.30
creating EKS cluster "my-eks-cluster" in "ap-southeast-1" region with managed nodes
will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
if you encounter any issues, check CloudFormation console or try `eksctl utils describe-stacks --region=ap-southeast-1 --cluster=my-eks-cluster`
Hubernetes API endpoint access will use default of (publicAccess=true, privateAccess=false) for cluster "my-eks-cluster" in "ap-southeast-1"
CloudWatch logging will not be enabled for cluster "my-eks-cluster" in "ap-southeast-1"
you can enable it with `eksctl utils update-cluster-logging --enable-types=[SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)] --region=ap-southeast-1 --cluster=my-eks-cluster`
default addons kube-proxy, coredns, vpc-cni were not specified, will install them as EKS addons
sequential tasks: ( create cluster control plane "my-eks-cluster",

```

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Fig 9.1. Creation of EKS Cluster 1

```

--cluster=my-eks-cluster'
2024-10-09 14:18:04 [!] default addons kube-proxy, coredns, vpc-cni were not specified, will install them as EKS addons
2024-10-09 14:18:04 []
2 sequential tasks: ( create cluster control plane "my-eks-cluster",
  2 sequential sub-tasks: (
    2 sequential sub-tasks: (
      1 task: ( create addons ),
      wait for control plane to become ready,
    ),
    create managed nodegroup "my-nodes",
  )
)
building cluster stack "eksctl-my-eks-cluster-cluster"
1 error(s) occurred and cluster hasn't been created properly, you may wish to check CloudFormation console
to cleanup resources, run `eksctl delete cluster --region=ap-southeast-1 --name=my-eks-cluster`'
creating CloudFormation stack "eksctl-my-eks-cluster-cluster": operation error CloudFormation: CreateStack, https response error StatusCode: 400, RequestID: b7e7478a-6735-4c5b-8933-9723ae0765c, AlreadyExistsException: Stack [eksctl-my-eks-cluster-cluster] already exists
Error: failed to create cluster "my-eks-cluster"
root@ip-172-31-41-180:~# eksctl delete cluster --region=ap-southeast-1 --name=my-eks-cluster
Error: unable to describe cluster control plane: operation error EKS: DescribeCluster, https response error StatusCode: 404, RequestID: e9e35aaaf-8b14-4117-b3a6-269cc052924, ResourceNotFoundException: No cluster found for name: my-eks-cluster.
root@ip-172-31-41-180:~# eksctl create cluster --name my-eks-cluster --region ap-southeast-1 --nodegroup-name my-nodes --nodes 2 --nodes-min 1 --nodes-max 2 --managed
eksctl version 0.191.0
using region ap-southeast-1
setting availability zones to [ap-southeast-1b ap-southeast-1a ap-southeast-1c]
subnets for ap-southeast-1b - public:192.168.0.0/19 private:192.168.96.0/19
subnets for ap-southeast-1a - public:192.168.32.0/19 private:192.168.128.0/19
subnets for ap-southeast-1c - public:192.168.64.0/19 private:192.168.160.0/19
nodegroup "my-nodes" will use "" [AmazonLinux2/1.30]
using Kubernetes version 1.30
creating EKS cluster "my-eks-cluster" in "ap-southeast-1" region with managed nodes
will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup

```

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Fig 9.2. Creation of EKS Cluster 2

```

aws Services Q forwarding rules X
CloudShell Feedback
© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

```

CloudWatch logs output:

```

2024-10-09 14:19:05 [i] subnets for ap-southeast-1c - public:192.168.64.0/19 private:192.168.160.0/19
2024-10-09 14:19:05 [i] nodegroup "my-nodes" will use "" [AmazonLinux2/1.30]
2024-10-09 14:19:05 [i] using Kubernetes version 1.30
2024-10-09 14:19:05 [i] creating EKS cluster "my-eks-cluster" in "ap-southeast-1" region with managed nodes
2024-10-09 14:19:05 [i] will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2024-10-09 14:19:05 [i] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=ap-southeast-1 --cluster=my-eks-cluster'
2024-10-09 14:19:05 [i] Kubernetes API endpoint access will use default of (publicAccess=true, privateAccess=false) for cluster "my-eks-cluster" in "ap-southeast-1"
2024-10-09 14:19:05 [i] CloudWatch logging will not be enabled for cluster "my-eks-cluster" in "ap-southeast-1"
2024-10-09 14:19:05 [i] you can enable it with 'eksctl utils update-cluster-logging --enable-types=(SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)) --region=ap-southeast-1 --cluster=my-eks-cluster'
2024-10-09 14:19:05 [i] default addons kube-proxy, coredns were not specified, will install them as EKS addons
2024-10-09 14:19:05 [i] 2 sequential tasks: ( create cluster control plane "my-eks-cluster",
  2 sequential sub-tasks: (
    1 task: ( create addons ),
    wait for control plane to become ready,
  ),
  create managed nodegroup "my-nodes",
)
2024-10-09 14:19:05 [i] building cluster stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:19:05 [!] 1 error(s) occurred and cluster hasn't been created properly, you may wish to check CloudFormation console
2024-10-09 14:19:05 [i] to cleanup resources, run 'eksctl delete cluster --region=ap-southeast-1 --name=my-eks-cluster'
2024-10-09 14:19:05 [X] creating CloudFormation stack "eksctl-my-eks-cluster-cluster": operation error CloudFormation: CreateStack, https response error StatusCo
[Err: 400, RequestID: e973ab66-4b14-4db7-af76-e39c03ae0574, AlreadyExistsException: Stack [eksctl-my-eks-cluster-cluster] already exists
Error: failed to create cluster "my-eks-cluster"
root@ip-172-31-41-180:~# eksctl create cluster --name my-eks-cluster --region ap-southeast-1 --nodegroup-name my-nodes --nodes 2 --nodes-min 1 --nodes-max 2 --man
aged
2024-10-09 14:21:52 [i] eksctl version 0.191.0
2024-10-09 14:21:52 [i] using region ap-southeast-1

```

Fig 9.3. Creation of EKS Cluster 3

```

aws Services Q forwarding rules X
CloudShell Feedback
© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

```

CloudWatch logs output:

```

2024-10-09 14:19:05 [i] building cluster stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:19:05 [!] 1 error(s) occurred and cluster hasn't been created properly, you may wish to check CloudFormation console
2024-10-09 14:19:05 [i] to cleanup resources, run 'eksctl delete cluster --region=ap-southeast-1 --name=my-eks-cluster'
2024-10-09 14:19:05 [X] creating CloudFormation stack "eksctl-my-eks-cluster-cluster": operation error CloudFormation: CreateStack, https response error StatusCo
[Err: 400, RequestID: e973ab66-4b14-4db7-af76-e39c03ae0574, AlreadyExistsException: Stack [eksctl-my-eks-cluster-cluster] already exists
Error: failed to create cluster "my-eks-cluster"
root@ip-172-31-41-180:~# eksctl create cluster --name my-eks-cluster --region ap-southeast-1 --nodegroup-name my-nodes --nodes 2 --nodes-min 1 --nodes-max 2 --man
aged
2024-10-09 14:21:52 [i] eksctl version 0.191.0
2024-10-09 14:21:52 [i] using region ap-southeast-1
2024-10-09 14:21:52 [i] setting availability zones to [ap-southeast-1a ap-southeast-1b ap-southeast-1c]
2024-10-09 14:21:52 [i] subnets for ap-southeast-1a - public:192.168.0.0/19 private:192.168.96.0/19
2024-10-09 14:21:52 [i] subnets for ap-southeast-1b - public:192.168.32.0/19 private:192.168.128.0/19
2024-10-09 14:21:52 [i] subnets for ap-southeast-1c - public:192.168.64.0/19 private:192.168.160.0/19
2024-10-09 14:21:52 [i] nodegroup "my-nodes" will use "" [AmazonLinux2/1.30]
2024-10-09 14:21:52 [i] using Kubernetes version 1.30
2024-10-09 14:21:52 [i] creating EKS cluster "my-eks-cluster" in "ap-southeast-1" region with managed nodes
2024-10-09 14:21:52 [i] will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2024-10-09 14:21:52 [i] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=ap-southeast-1 --cluster=my-eks-cluster'
2024-10-09 14:21:52 [i] Kubernetes API endpoint access will use default of (publicAccess=true, privateAccess=false) for cluster "my-eks-cluster" in "ap-southeast-1"
2024-10-09 14:21:52 [i] CloudWatch logging will not be enabled for cluster "my-eks-cluster" in "ap-southeast-1"
2024-10-09 14:21:52 [i] you can enable it with 'eksctl utils update-cluster-logging --enable-types=(SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)) --region=ap-southeast-1 --cluster=my-eks-cluster'
2024-10-09 14:21:52 [i] default addons coredns, kube-proxy, vpc-cni were not specified, will install them as EKS addons
2024-10-09 14:21:52 [i] 2 sequential tasks: ( create cluster control plane "my-eks-cluster",
  2 sequential sub-tasks: (
    1 task: ( create addons ),
    wait for control plane to become ready,
  ),
)
2024-10-09 14:21:52 [i] CloudShell Feedback

```

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

```

2024-10-09 14:21:52 [i] building cluster stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:21:52 [i] deploying stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:22:22 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:22:52 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:23:53 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:24:53 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:25:53 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:26:53 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:27:53 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:28:53 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:29:53 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:30:53 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"
2024-10-09 14:30:53 [i] creating addon
2024-10-09 14:30:54 [i] successfully created addon
2024-10-09 14:30:54 [i] recommended policies were found for "vpc-cni" addon, but since OIDC is disabled on the cluster, eksctl cannot configure the requested permissions; the recommended way to provide IAM permissions for "vpc-cni" addon is via pod identity associations; after addon creation is completed, add all recommended policies to the config file, under 'addon.PodIdentityAssociations', and run 'eksctl update addon'
2024-10-09 14:30:54 [i] creating addon
2024-10-09 14:30:54 [i] successfully created addon
2024-10-09 14:30:55 [i] creating addon
2024-10-09 14:30:55 [i] successfully created addon
2024-10-09 14:32:55 [i] building managed nodegroup stack "eksctl-my-eks-cluster-nodegroup-my-nodes"
2024-10-09 14:32:55 [i] deploying stack "eksctl-my-eks-cluster-nodegroup-my-nodes"
2024-10-09 14:32:55 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-my-nodes"
2024-10-09 14:33:25 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-my-nodes"
2024-10-09 14:34:07 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-my-nodes"
2024-10-09 14:35:20 [i] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-my-nodes"
2024-10-09 14:35:20 [i] waiting for the control plane to become ready
2024-10-09 14:35:20 [✓] saved kubeconfig as "/etc/kubernetes/admin.conf"
2024-10-09 14:35:20 [i] no tasks
2024-10-09 14:35:20 [✓] all EKS cluster resources for "my-eks-cluster" have been created

```

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

```

2024-10-09 14:35:21 [i] nodegroup "my-nodes" has 2 node(s)
2024-10-09 14:35:21 [i] node "ip-192-168-29-114.ap-southeast-1.compute.internal" is ready
2024-10-09 14:35:21 [i] node "ip-192-168-82-192.ap-southeast-1.compute.internal" is ready
2024-10-09 14:35:21 [i] waiting for at least 1 node(s) to become ready in "my-nodes"
2024-10-09 14:35:21 [i] nodegroup "my-nodes" has 2 node(s)
2024-10-09 14:35:21 [i] node "ip-192-168-29-114.ap-southeast-1.compute.internal" is ready
2024-10-09 14:35:21 [i] node "ip-192-168-82-192.ap-southeast-1.compute.internal" is ready
2024-10-09 14:35:21 [✓] created 1 managed nodegroup(s) in cluster "my-eks-cluster"
2024-10-09 14:35:22 [i] kubectl command should work with "/etc/kubernetes/admin.conf", try "kubectl --kubeconfig=/etc/kubernetes/admin.conf get nodes"
2024-10-09 14:35:22 [✓] EKS cluster "my-eks-cluster" in "ap-southeast-1" region is ready

```

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.

**11.1.5. 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.asc \ 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,

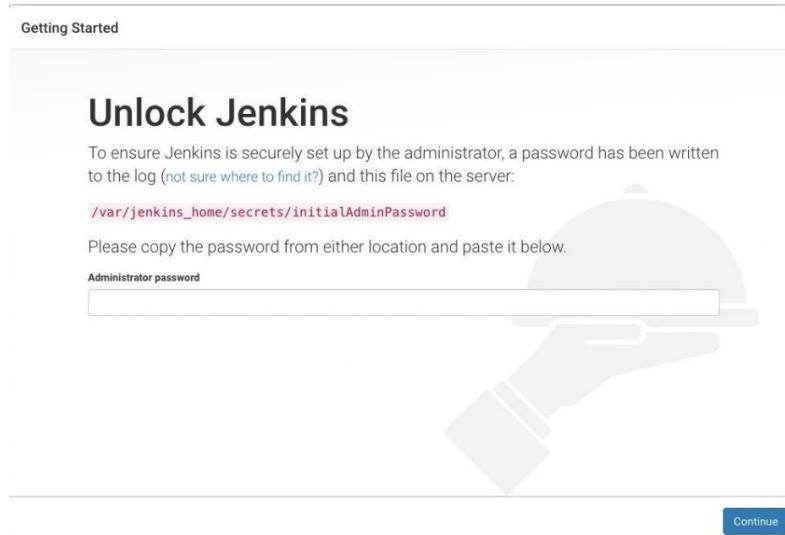


Fig 10.1. Unlock screen of Jenkins

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:

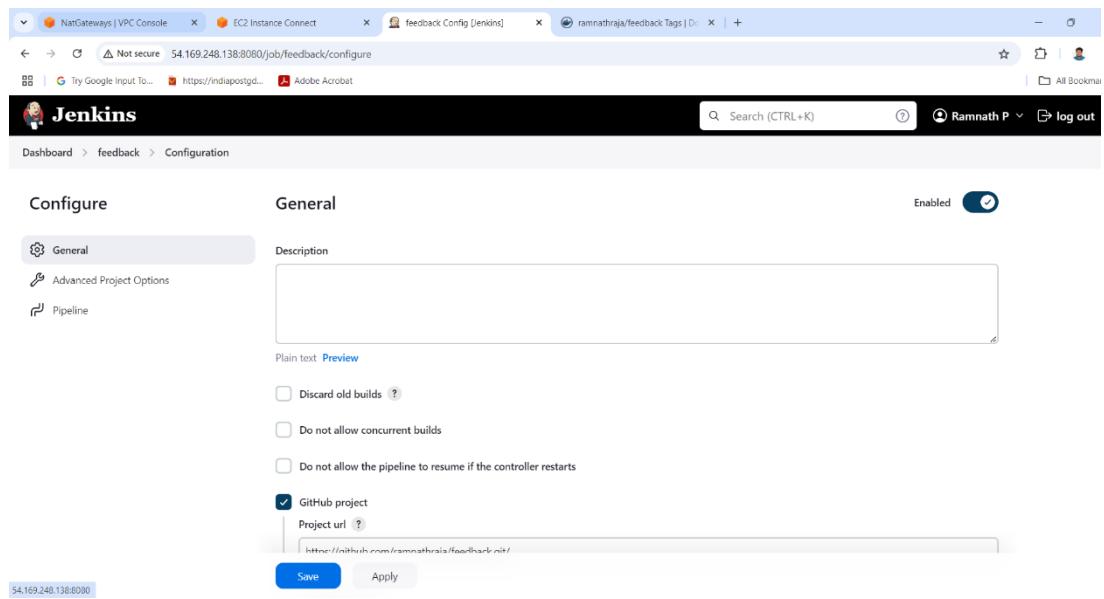


Fig 10.2. Configuration of Jenkins 1

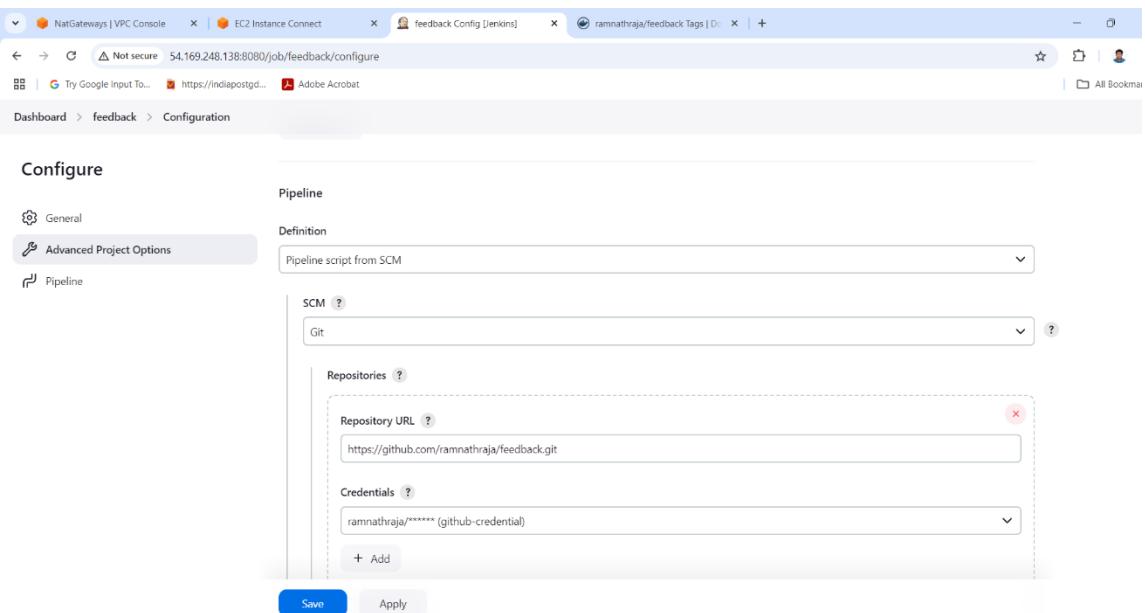


Fig 10.3. Configuration of Jenkins 2

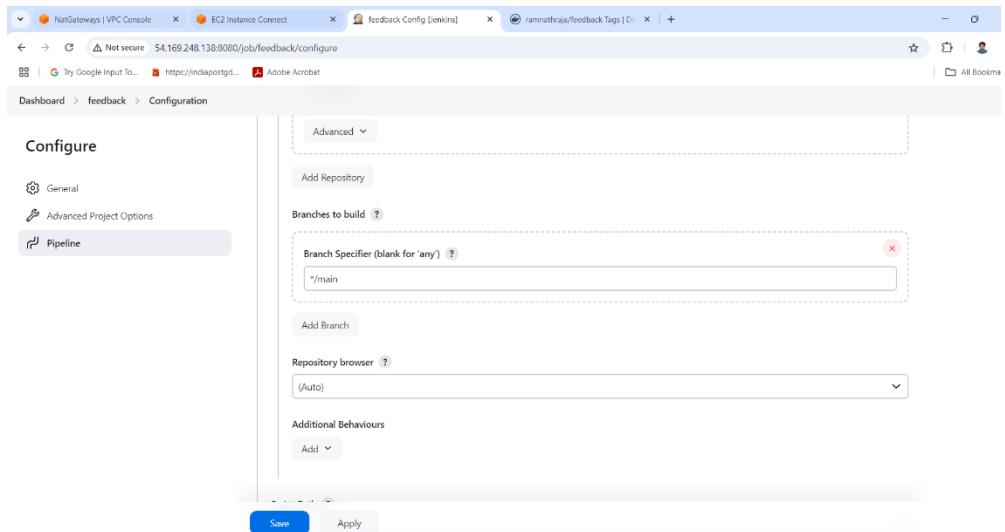


Fig 10.4. Configuration of Jenkins 3

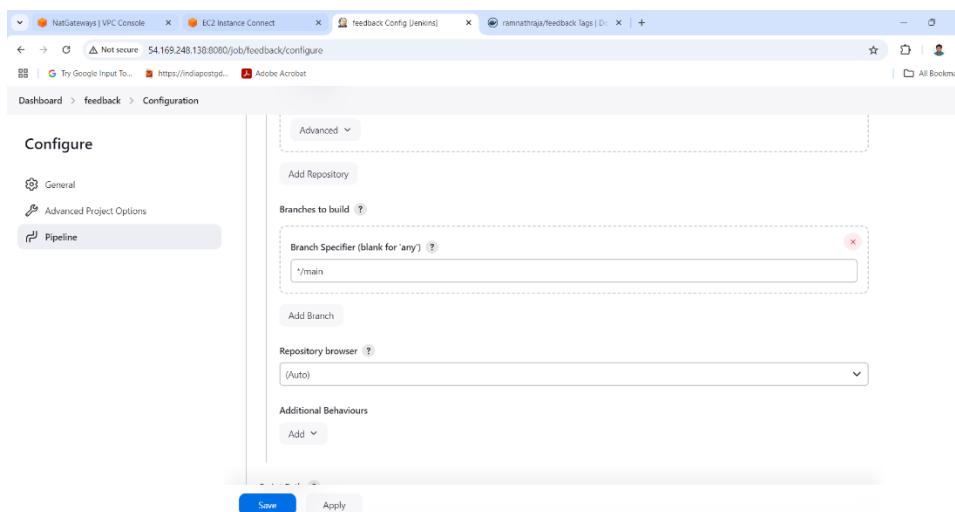


Fig 10.5. Configuration of Jenkins 4

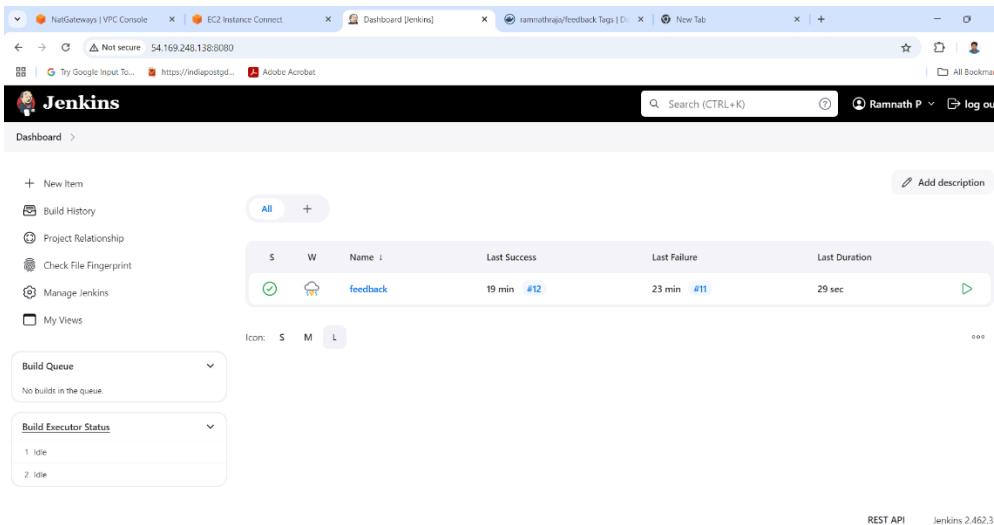


Fig 10.6. Configuration of Jenkins 5

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.

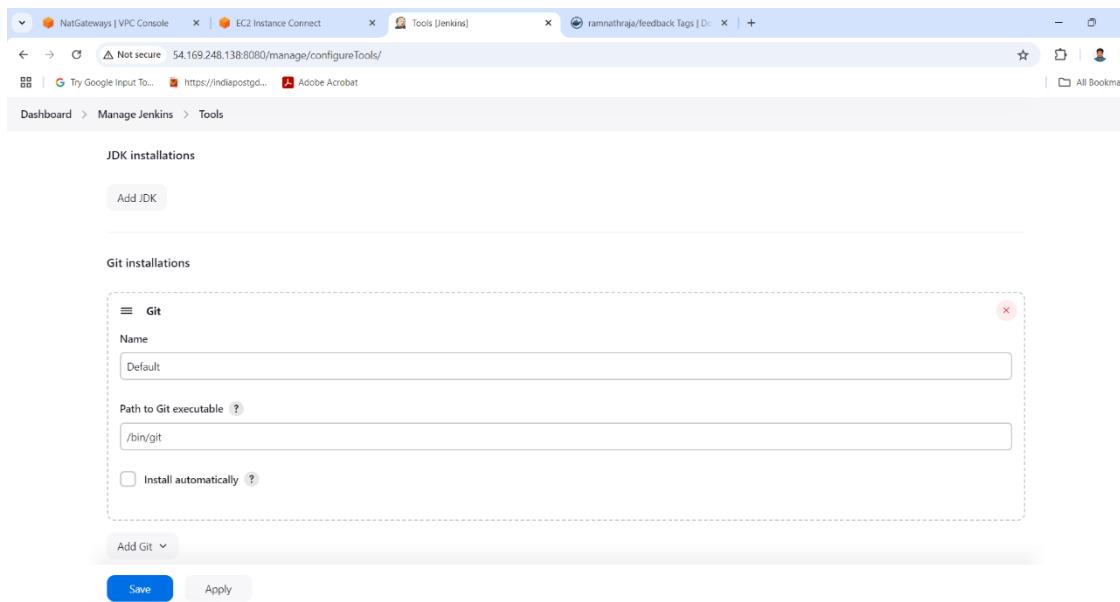


Fig 10.7. Configuration of Gobal Tool Configuration

## 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.

The screenshot shows the Jenkins Credentials management page. At the top, there are tabs for 'NatGateways | VPC Console', 'EC2 Instance Connect', 'Jenkins - Credentials [jenkins]', and 'ramnathraja/feedback Tags | Do...'. Below the tabs, the address bar shows 'Not secure 54.169.248.138:8080/manage/credentials/'. The main content area has a header 'Credentials' with columns: T, P, Store, Domain, ID, and Name. There are three entries:

T	P	Store	Domain	ID	Name
		System	(global)	github-credential	ramnathraja/******** (github-credential)
		System	(global)	docker-hub-credentials	ramnathraja/******** (docker-hub-credentials)
		System	(global)	aws-eks-credential	AKIAVOJ7YEMTZRGBWDTB

Below this, a section titled 'Stores scoped to Jenkins' shows a single entry:

P	Store	Domains
	System	(global)

At the bottom left, there are icons for 'Icon: S M L'.

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.

The screenshot shows the Jenkins interface for managing nodes. The top navigation bar includes tabs for 'NatGateways | VPC Console', 'EC2 Instance Connect', 'Nodes [Jenkins]', and 'ramnathraja/feedback Tags | Details'. The main title is 'Jenkins' with a subtitle 'Nodes'. Below the title, there are links for 'Dashboard', 'Manage Jenkins', and 'Nodes'. A search bar and user profile are also present. The main content area is titled 'Nodes' and contains a table with the following data:

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	Built-In Node	Linux (amd64)	In sync	1.46 GiB	0 B	1.46 GiB	0ms
		last checked	54 min	54 min	54 min	54 min	54 min

On the left, there are sections for 'Build Queue' (empty) and 'Build Executor Status' (1 Idle, 2 Idle). A legend at the bottom indicates icons for Small (S), Medium (M), and Large (L) nodes. The bottom right corner shows 'REST API' and 'Jenkins 2.462.3'.

Fig 10.9. Node in Jenkins

## Starting of the Jenkins Pipeline:

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

The screenshot shows the Jenkins dashboard with the 'feedback' pipeline job listed in the build queue. The job has a green checkmark icon, indicating it is successful. It was last run 19 minutes ago and took 12 seconds. The build queue section shows 'No builds in the queue'. The build executor status section shows 1 idle executor.

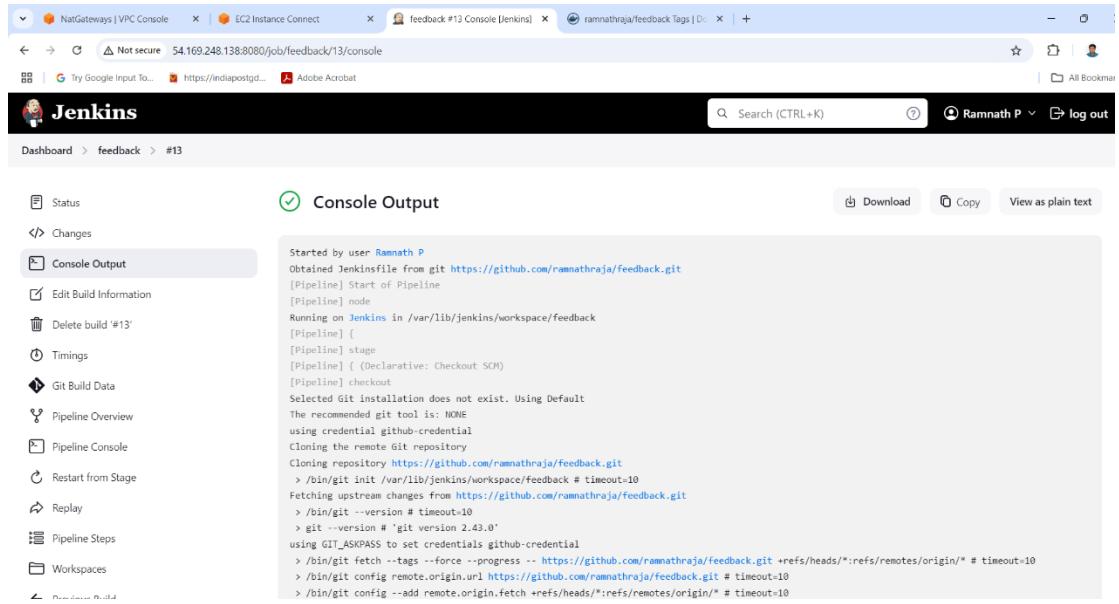
Fig 10.10. Feedback pipeline job

We can see the Success build in the Build Queue

The screenshot shows the Jenkins job details for 'feedback #13'. The build status is 'Success' (green checkmark). It was started 3 minutes 51 seconds ago and took 29 seconds. The build information includes the user 'Rammath P' and the revision 'abb43ab1a12336ff2bcf7840467dd9a040862fe7'. The repository is 'https://github.com/rammathraja/feedback.git'. The build summary lists '18 ms waiting', '29 sec build duration', and '29 sec total from scheduled to completion'. The left sidebar shows various Jenkins management options like Status, Changes, Console Output, and Pipeline Overview.

Fig 10.11. Successful Built

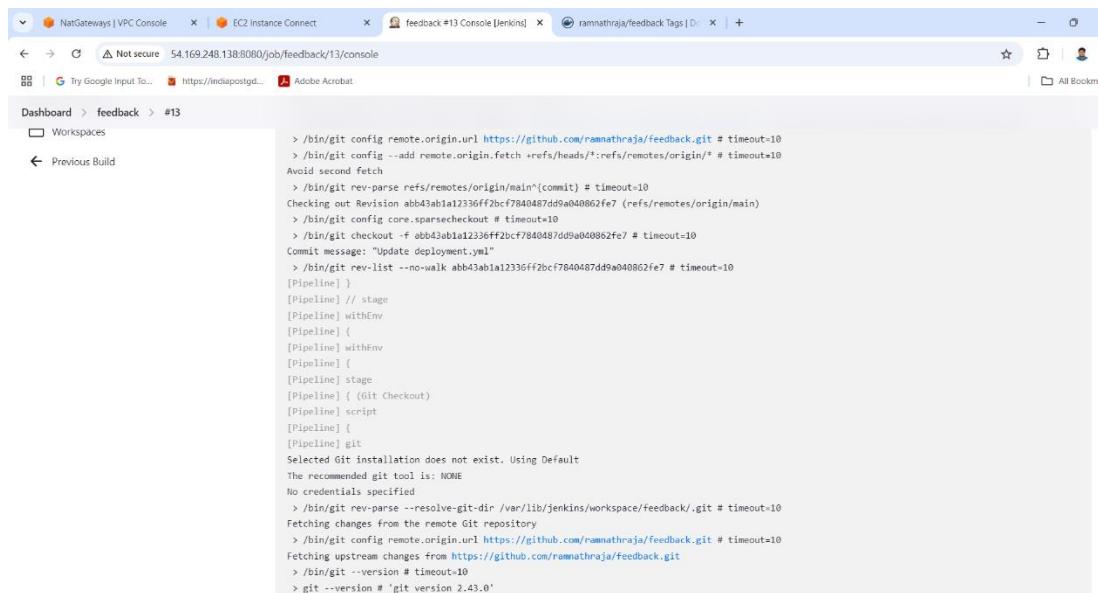
We can see the output of the build in the console output tab,



The screenshot shows the Jenkins interface with the 'Console Output' tab selected for build #13. The log output is as follows:

```
Started by user Rammath P
Obtained Jenkinsfile from git https://github.com/ramnathraja/feedback.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/feedback
[Pipeline] {
[Pipeline] stage
[Pipeline] {
  (Declarative: Checkout SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
using credential github-credential
Cloning the remote Git repository
Cloning repository https://github.com/ramnathraja/feedback.git
> /bin/git init /var/lib/jenkins/workspace/feedback # timeout=10
Fetching upstream changes from https://github.com/ramnathraja/feedback.git
> /bin/git --version # timeout=10
> git --version # 'git' version 2.43.0'
using GIT_ASKPASS to set credentials github-credential
> /bin/git fetch --tags --force --progress -- https://github.com/ramnathraja/feedback.git +refs/heads/*:refs/remotes/origin/*
> /bin/git config remote.origin.url https://github.com/ramnathraja/feedback.git # timeout=10
> /bin/git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
```

Fig 10.12. Successful Built console output 1



The screenshot shows the Jenkins interface with the 'Console Output' tab selected for build #13. The log output is as follows:

```
> /bin/git config remote.origin.url https://github.com/ramnathraja/feedback.git # timeout=10
> /bin/git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
> /bin/git rev-parse refs/remotes/origin/main^(commit) # timeout=10
Checking out Revision abbd3ab1a12336ff2bcf7840487dd9a040862fe7 (refs/remotes/origin/main)
> /bin/git config core.sparsecheckout # timeout=10
> /bin/git checkout -f abbd3ab1a12336ff2bcf7840487dd9a040862fe7 # timeout=10
Commit message: "Update deployment.yml"
> /bin/git rev-list --no-walk abbd3ab1a12336ff2bcf7840487dd9a040862fe7 # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] {
  (Git Checkout)
[Pipeline] script
[Pipeline] {
[Pipeline] git
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
> /bin/git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/feedback/.git # timeout=10
Fetching changes from the remote Git repository
> /bin/git config remote.origin.url https://github.com/ramnathraja/feedback.git # timeout=10
Fetching upstream changes from https://github.com/ramnathraja/feedback.git
> /bin/git --version # timeout=10
> git --version # 'git' version 2.43.0'
```

Fig 10.13. Successful Built console output 2

```

> /bin/git --version # timeout=10
> git --version # 'git' version 2.43.0'
> /bin/git fetch --tags --force --progress -- https://github.com/ramnathraja/feedback.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> /bin/git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision abb43ab1a12330ff2bcf7840487dd9a040862fe7 (refs/remotes/origin/main)
> /bin/git config core.sparsecheckout # timeout=10
> /bin/git checkout -f abb43ab1a12330ff2bcf7840487dd9a040862fe7 # timeout=10
> /bin/git branch -a -v --no-abbrev # timeout=10
> /bin/git checkout -b main abb43ab1a12330ff2bcf7840487dd9a040862fe7 # timeout=10
Commit message: "Update deployment.yml"
[Pipeline]
[Pipeline] // script
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Building Docker Image)
[Pipeline] script
[Pipeline] {
[Pipeline] isUnix
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
+ docker build -t ramnathraja/feedback:13 .
#0 building with "default" instance using docker driver

#1 [internal] load build definition from Dockerfile
#1 transferring dockerfile: 463B done
#1 DONE 0.0s

```

Fig 10.14. Successful Built console output 3

```

#1 DONE 0.0s

#2 [internal] load metadata for docker.io/library/node:14
#2 DONE 1.0s

#3 [internal] load .dockerignore
#3 transferring context: 2B done
#3 DONE 0.0s

#4 [1/5] FROM docker.io/library/node:14@sha256:a150d3b9b4e3fa813fa6c8c590b0f0a060e015ad4e59bbce5744d2f6fd8461aa
#4 DONE 0.0s

#5 [internal] load build context
#5 transferring context: 2.65MB 0.0s done
#5 DONE 0.1s

#6 [2/5] WORKDIR /app
#6 CACHED

#7 [3/5] COPY package*.json ./
#7 CACHED

#8 [4/5] RUN npm install
#8 CACHED

#9 [5/5] COPY . /app
#9 DONE 0.1s

#10 exporting to image
#10 exporting layers 0.0s done

```

Fig 10.15. Successful Built console output 4

```

#10 exporting to image
#10 exporting layers 0.0s done
#10 writing image sha256:6fd43259916872571b617a414e60189d5bd52c050dd5ae6e69b2acaa3013b69 done
#10 naming to docker.io/ramnathraja/feedback:13 done
#10 DONE 0.1s
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Push Image To Docker Hub)
[Pipeline] script
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] withDockerRegistry
$ docker login -u ramnathraja -p ***** https://registry.hub.docker.com
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /var/lib/jenkins/workspace/feedback@tmp/d45337d-b6b1-46d4-bcd5-b1c3f0417a01/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
[Pipeline] {
[Pipeline] isUnix
[Pipeline] withEnv
[Pipeline] {

```

Fig 10.16. Successful Built console output 5

```

https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
[Pipeline] {
[Pipeline] isUnix
[Pipeline] withEnv
[Pipeline] {
[Pipeline] {
[Pipeline] sh
+ docker tag ramnathraja/feedback:13 registry.hub.docker.com/ramnathraja/feedback:13
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] isUnix
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
+ docker push registry.hub.docker.com/ramnathraja/feedback:13
The push refers to repository [registry.hub.docker.com/ramnathraja/feedback]
ce04ba13d067: Preparing
b099e22288a4: Preparing
5f70bf1ba086: Preparing
21868587f31e: Preparing
0df5f1a15e5d: Preparing
3c777d951da2: Preparing
f8a91dd5fc84: Preparing
cb8127abde5: Preparing
e01ad454893a0: Preparing
c456680adde37: Preparing
fe0fb3b4a0f: Preparing
f1186c5961f2: Preparing

```

Fig 10.17. Successful Built console output 6

```
fe0fb3ab4a0f: Preparing
f1186e5061f2: Preparing
b2dbfa7477754: Preparing
3c777d951de2: Waiting
f8a91d05fc84: Waiting
cb81227abde5: Waiting
e01a454893a9: Waiting
c45660adde37: Waiting
fe0fb3ab4a0f: Waiting
f1186e5061f2: Waiting
b2dbfa7477754: Waiting
5f70bf18a086: Layer already exists
0d5f5a015e5d: Layer already exists
21808587f71c: Layer already exists
b099a22288a4: Layer already exists
f8a91d05fc84: Layer already exists
3c777d951de2: Layer already exists
cb81227abde5: Layer already exists
e01a454893a9: Layer already exists
c45660adde37: Layer already exists
f1186e5061f2: Layer already exists
fe0fb3ab4a0f: Layer already exists
b2dbfa7477754: Layer already exists
ce49da13d867: Pushed
13: digest: sha256:194ac196e7a286669b854f6e4b5271d3bc12562cc6c1b6f2493e2316c6639ca0 size: 3045
[Pipeline]
[Pipeline] // withEnv
[Pipeline] isUnix
[Pipeline] withEnv
[Pipeline] /
```

Fig 10.18. Successful Built console output 7

```
[Pipeline] isUnix
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
+ docker tag rammathraja/feedback:13 registry.hub.docker.com/rammathraja/feedback:latest
[Pipeline]
[Pipeline] // withEnv
[Pipeline] isUnix
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
+ docker push registry.hub.docker.com/rammathraja/feedback:latest
The push refers to repository [registry.hub.docker.com/rammathraja/feedback]
ce49da13d867: Preparing
b099a22288a4: Preparing
5f70bf18a086: Preparing
21808587f71c: Preparing
0d5f5a015e5d: Preparing
3c777d951de2: Preparing
f8a91d05fc84: Preparing
cb81227abde5: Preparing
e01a454893a9: Preparing
c45660adde37: Preparing
fe0fb3ab4a0f: Preparing
f1186e5061f2: Preparing
b2dbfa7477754: Preparing
3c777d951de2: Waiting
f8a91d05fc84: Waiting
cb81227abde5: Waiting
e01a454893a9: Waiting
```

Fig 10.19. Successful Built console output 8

```
curl -XPOST https://54.169.248.138:8080/job/feedback/13/console
cb81227ab0de5: Waiting
e01a4548939d: Waiting
c45600adde37: Waiting
fe0fb3abab0a0: Waiting
f1186e5061f: Waiting
b2dba7477754: Waiting
b099a2228d4: Layer already exists
21808597f31c: Layer already exists
0d5f5a019c5d: Layer already exists
5f70bf18a086: Layer already exists
ce49da13d867: Layer already exists
e01a4548939d: Layer already exists
3c777d0512e2: Layer already exists
#8a01d5f5c84: Layer already exists
cb81227ab0de5: Layer already exists
c45600adde37: Layer already exists
f1186e5061f: Layer already exists
fe0fb3abab0a0: Layer already exists
b2dba7477754: Layer already exists
latest: digest: sha256:194ac196e7a286669b854f6e4b5231d3bc12562cc6c1b6f2493e2316c6639caf size: 3045
[Pipeline]
[Pipeline] /
[Pipeline] /
[Pipeline] // withEnv
[Pipeline] /
[Pipeline] // withDockerRegistry
[Pipeline] /
[Pipeline] // withEnv
[Pipeline] /
[Pipeline] // script
[Pipeline] /
```

Fig 10.20. Successful Built console output 9

```
[Pipeline] // script
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy to Kubernetes)
[Pipeline] withCredentials
Masking supported pattern matches of $AWS_ACCESS_KEY_ID or $AWS_SECRET_ACCESS_KEY
[Pipeline]
[Pipeline] {
[Pipeline] script
[Pipeline] {
[Pipeline] sh
+ aws eks update-kubeconfig --name my-eks-cluster --region ap-southeast-1
Updated context arn:aws:eks:ap-southeast-1:37433278103:cluster/my-eks-cluster in /var/lib/jenkins/kubeconfig
[Pipeline] sh
+ kubectl apply -f deployment.yaml
deployment.apps/demo-deployment unchanged
service/demo-service unchanged
poddisruptionbudget.policy/demo-pdb configured
[Pipeline]
[Pipeline] // script
[Pipeline]
[Pipeline] // withCredentials
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] cleanks
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
```

Fig 10.21. Successful Built console output 10

```

poddisruptionbudget.policy/demo-pdb configured
[Pipeline]
[Pipeline] // script
[Pipeline]
[Pipeline] // withCredentials
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] {
  [Pipeline] (Declarative: Post Actions)
  [Pipeline] cleanWs
  [WS-CLEANUP] Deleting project workspace...
  [WS-CLEANUP] Deferred wipeout is used...
  [WS-CLEANUP] done
  [Pipeline]
  [Pipeline] // stage
  [Pipeline]
  [Pipeline] // withEnv
  [Pipeline]
  [Pipeline] // stage
  [Pipeline]
  [Pipeline] end of Pipeline
}
Finished: SUCCESS

```

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.

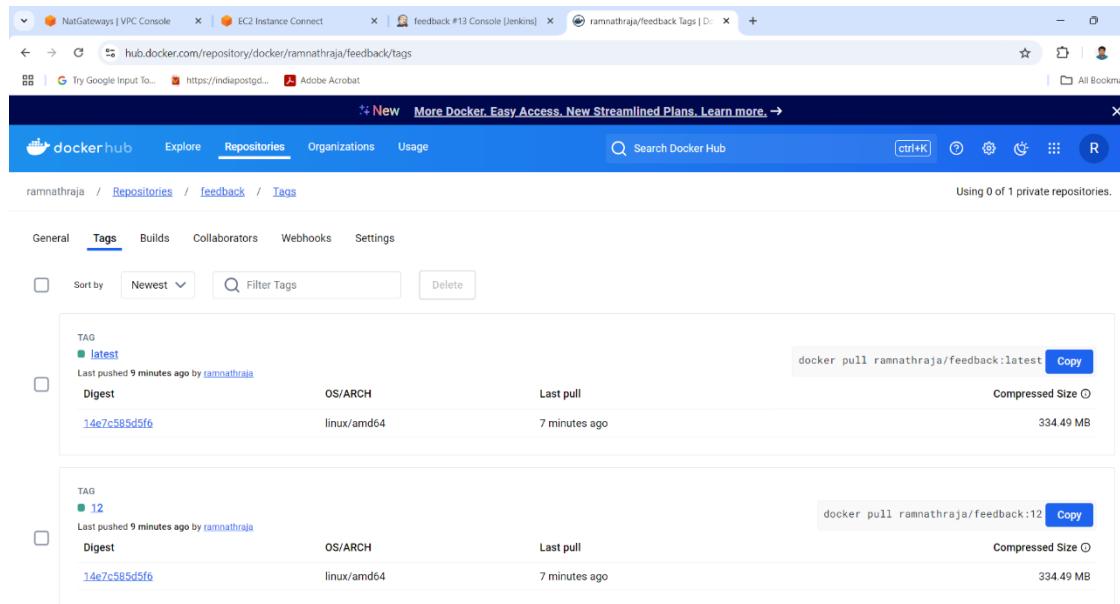


Fig 11. Docker Hub Registry

## 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.

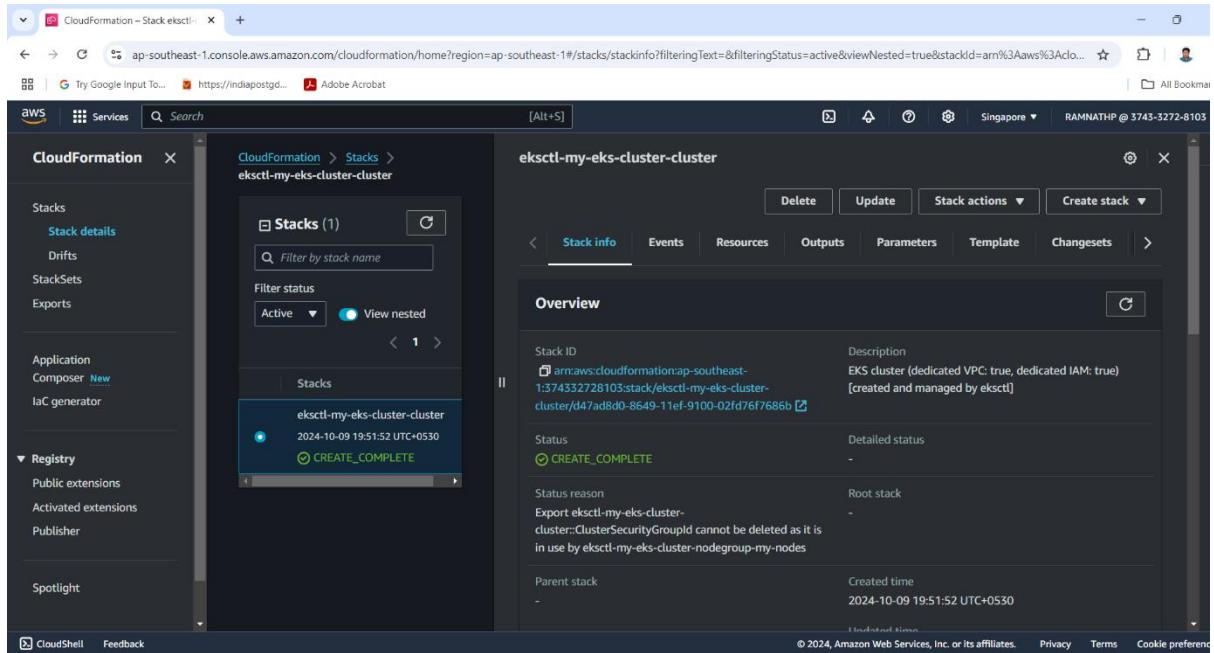


Fig 12. My-eks-cluster

## 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.

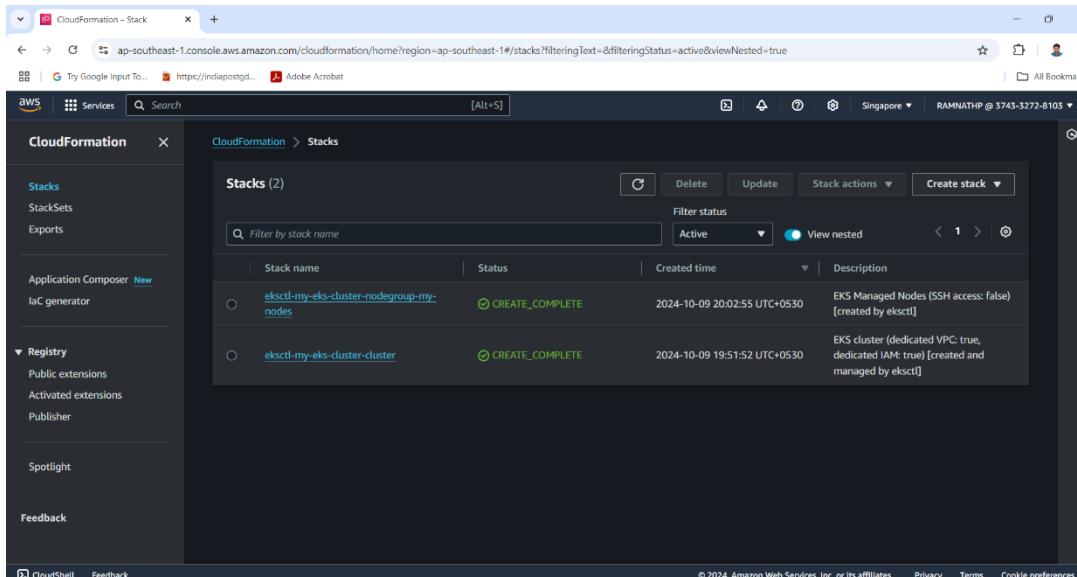


Fig 13. Cloud Formation Stack

## 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.

```

root@ip-172-31-41-180:~# kubectl get all
NAME                           READY   STATUS    RESTARTS   AGE
pod/demo-deployment-69486b5467-b7tmp   0/1     CrashLoopBackOff   9 (78s ago)   22m
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP          PORT(S)          AGE
service/demo-service   LoadBalancer   10.100.250.113   aae3bac15f3724e709935f057c5dcc6b-1149757469.ap-southeast-1.elb.amazonaws.com   8081:30008/TCP   22m
service/kubernetes   ClusterIP    10.100.0.1       <none>           443/TCP         78m
NAME             READY   UP-TO-DATE   CURRENT   READY   AGE
deployment.apps/demo-deployment   0/1     1           1         0      22m
replicaset.apps/demo-deployment-69486b5467   1           1         0      0      22m
root@ip-172-31-41-180:~# 

```

The screenshot shows a terminal window within an AWS CloudShell interface. The user has run the command `kubectl get all`. The output displays information about pods, services, and replicatesets. A pod named `demo-deployment-69486b5467-b7tmp` is shown in a `CrashLoopBackOff` state. Two services are listed: one for the application (`demo-service`) and one for Kubernetes (`kubernetes`). The application service has an external IP and is accessible via port 8081. A replicaset named `demo-deployment-69486b5467` is also listed.

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.