Jenkins Pipeline 3 - CICD with Terraform

CI -> Continuous Integration

CD -> Continuous Deployment or Delivery

CI Job:

Multiple stages will be there

Pipeline1: Git Clone -> Maven Build -> Docker Image -> Push Image -> Deployment

CD Job:

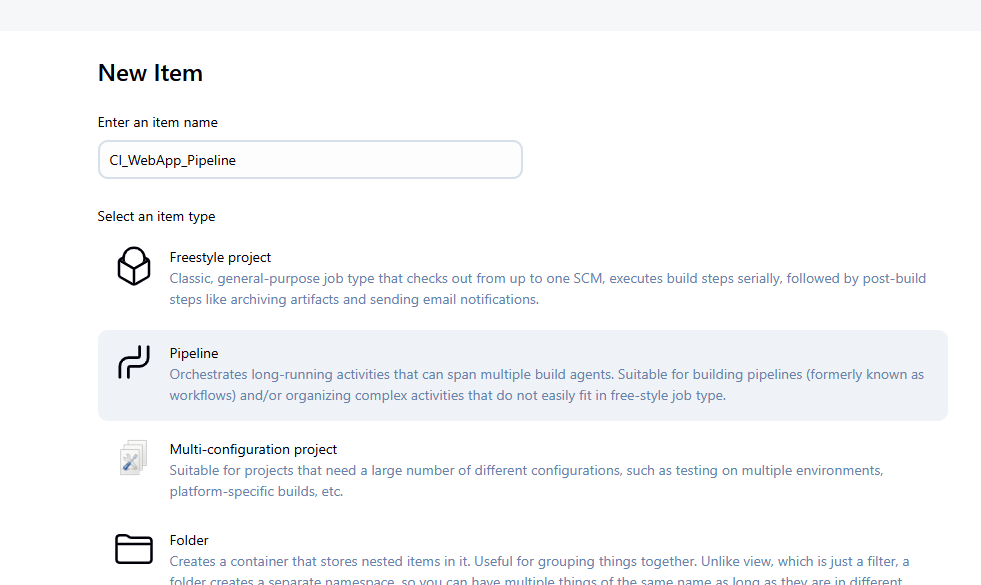
Pipeline2: Git Clone (to clone K8s manifest file and if we are writing manifest file in pipeline then need not clone) -> K8s Deployment

If Pipeline1 is successful, it should automatically trigger the CD job

Starting EKS clusters in EKS-host VM

eksctl create cluster --name my-eks-cluster --region ca-central-1 --node-type t2.medium --zones ca-central-1a,ca-central-1b

New Item



This repo has both manifest and Dockerfile

pipeline {

agent any

environment {

IMAGE\_NAME = "CI-web-app"

DOCKER\_TAG = "latest"

}

tools {

maven "maven-3.9.10"

}

stages {

stage('git clone') {

steps {

git branch: 'main', url: 'https://github.com/Haider7214/WebAppMaven.git'

}

}

stage('maven build') {

steps {

sh 'mvn clean package'

}

}

stage('Build Docker Image') {

steps {

script {

writeFile file: 'Dockerfile', text: '''

# Use an official Tomcat base image

FROM tomcat:latest

LABEL maintainer="DemoDockerfile"

# Remove default webapps

RUN rm -rf /usr/local/tomcat/webapps/\*

# Copy WAR to Tomcat webapps

COPY target/\*.war /usr/local/tomcat/webapps/ROOT.war

# Expose port

EXPOSE 8080

'''

echo "✅ Dockerfile generated"

sh "docker build -t ${IMAGE\_NAME}:${DOCKER\_TAG} ."

}

}

}

stage('Docker push') {

steps {

withCredentials([string(credentialsId: 'Sai-Docker-Pwd', variable: 'Docker\_Hub\_PWD\_New')]) {

sh 'docker login -u saidocker567 -p ${Docker\_Hub\_PWD\_New}'

sh 'docker tag ${IMAGE\_NAME}:${DOCKER\_TAG} saidocker567/${IMAGE\_NAME}:${DOCKER\_TAG}'

sh 'docker push saidocker567/${IMAGE\_NAME}:${DOCKER\_TAG}'

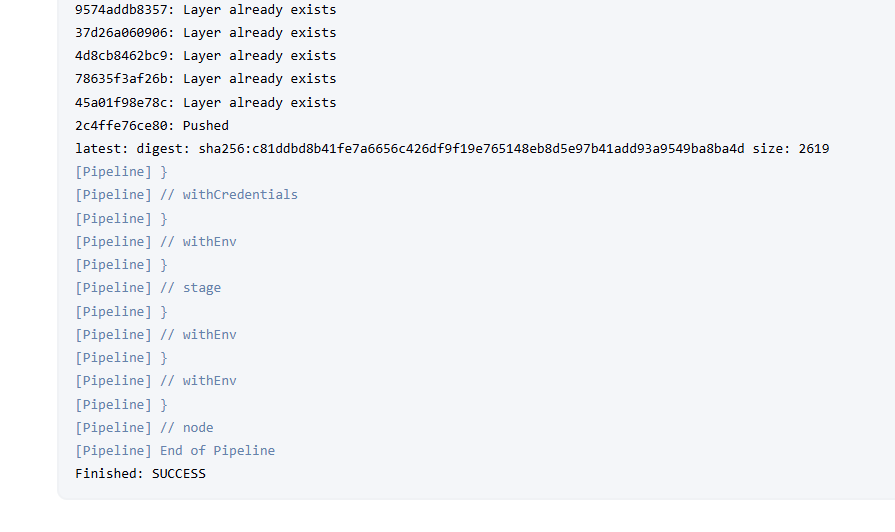
}

}

}

}

}



Change the image name

pipeline {

agent any

environment {

IMAGE\_NAME = "ci-web-app"

DOCKER\_TAG = "latest"

}

tools {

maven "maven-3.9.10"

}

stages {

stage('git clone') {

steps {

git branch: 'main', url: 'https://github.com/Haider7214/WebAppMaven.git'

}

}

stage('maven build') {

steps {

sh 'mvn clean package'

}

}

stage('Build Docker Image') {

steps {

script {

writeFile file: 'Dockerfile', text: '''

# Use an official Tomcat base image

FROM tomcat:latest

LABEL maintainer="DemoDockerfile"

# Remove default webapps

RUN rm -rf /usr/local/tomcat/webapps/\*

# Copy WAR to Tomcat webapps

COPY target/\*.war /usr/local/tomcat/webapps/ROOT.war

# Expose port

EXPOSE 8080

'''

echo "✅ Dockerfile generated"

sh "docker build -t ${IMAGE\_NAME}:${DOCKER\_TAG} ."

}

}

}

stage('Docker push') {

steps {

withCredentials([string(credentialsId: 'Sai-Docker-Pwd', variable: 'Docker\_Hub\_PWD\_New')]) {

sh 'docker login -u saidocker567 -p ${Docker\_Hub\_PWD\_New}'

sh 'docker tag ${IMAGE\_NAME}:${DOCKER\_TAG} saidocker567/${IMAGE\_NAME}:${DOCKER\_TAG}'

sh 'docker push saidocker567/${IMAGE\_NAME}:${DOCKER\_TAG}'

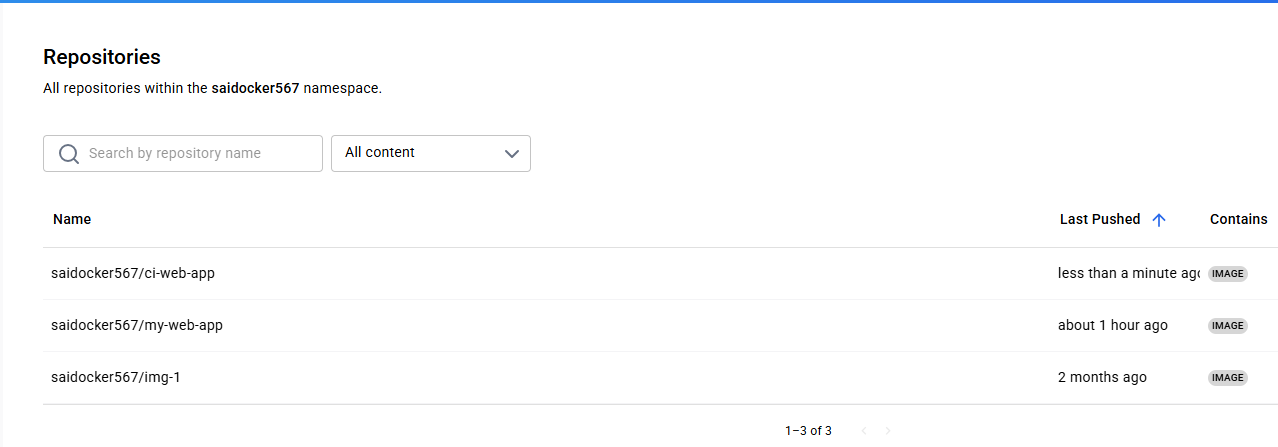
}

}

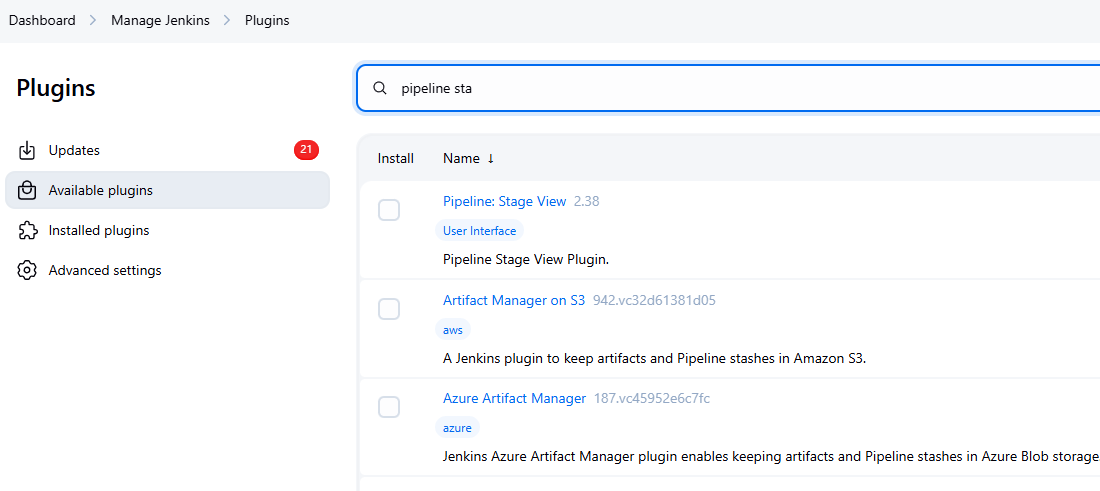
}

}

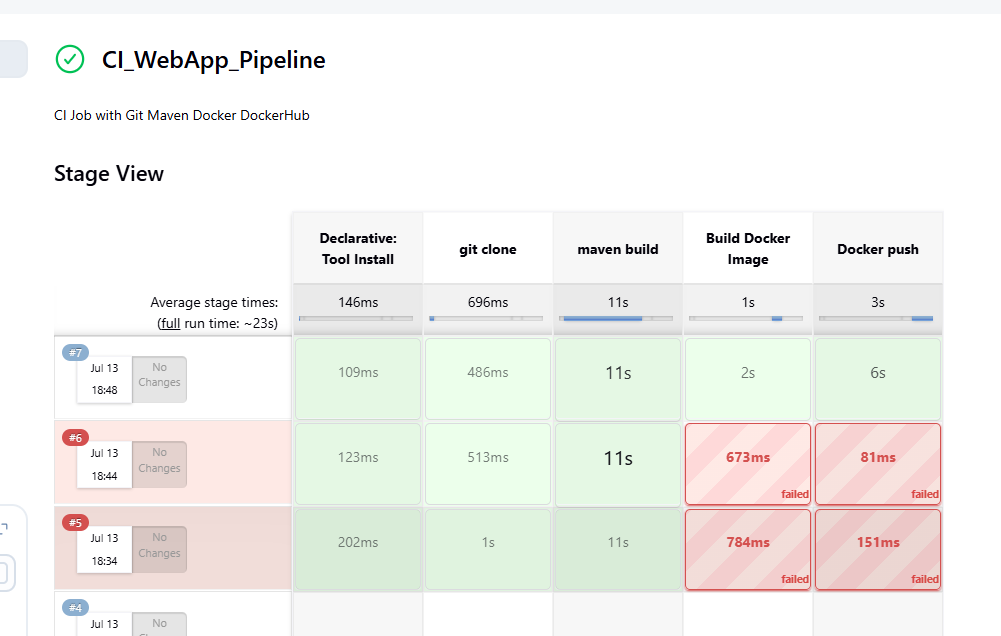
}



To trigger CD, we need Pipeline Stage



Different view due to Pipeline Stage View Plugin



CD as another pipeline

git clone ( to clone k8s manifest file and if we are writing manifest file in pipeline then need not to clone)

k8s deployment

pipeline {

agent any

tools {

maven "maven-3.9.10"

}

stages {

stage('git clone') {

steps {

git branch: 'main', url: 'https://github.com/Haider7214/WebAppMaven.git'

}

}

stage('k8s - deployment') {

steps {

sh 'kubectl apply -f k8s-deployment.yaml'

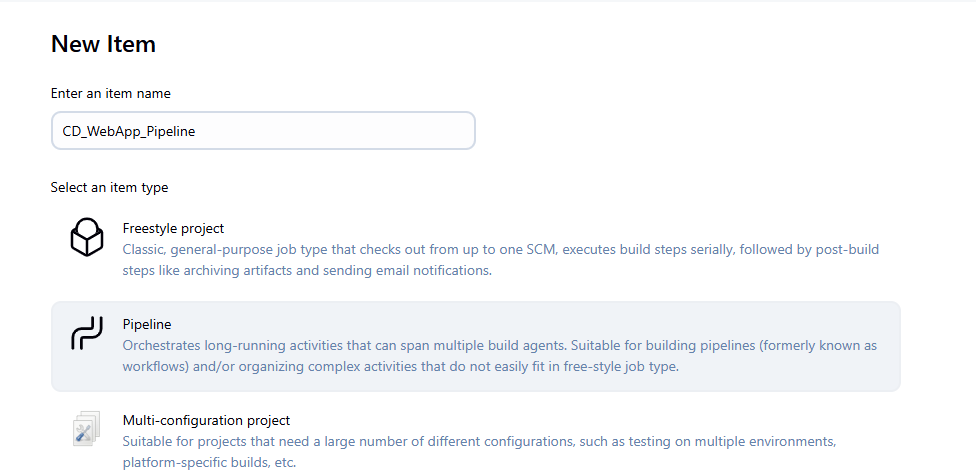
}

}

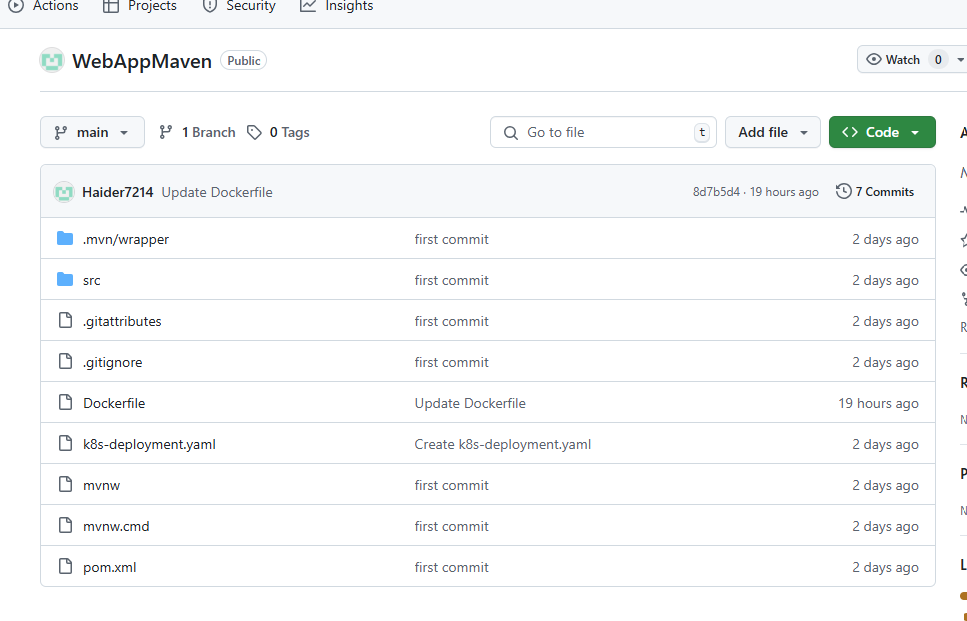
}

}

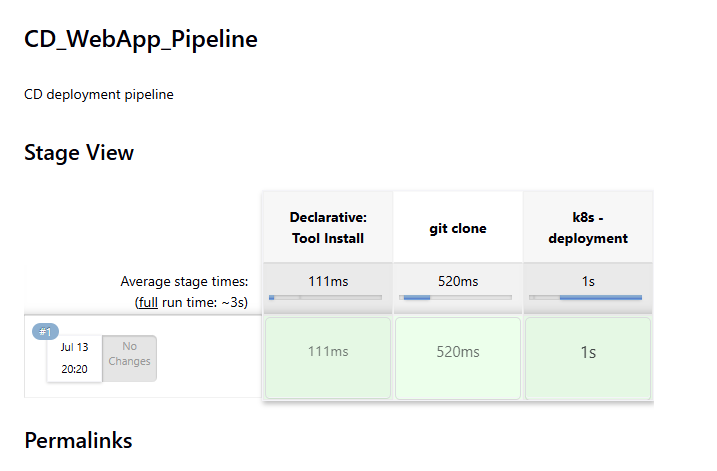
35:44



We see k8s-deployment.yaml and Dockerfile are there in the repo



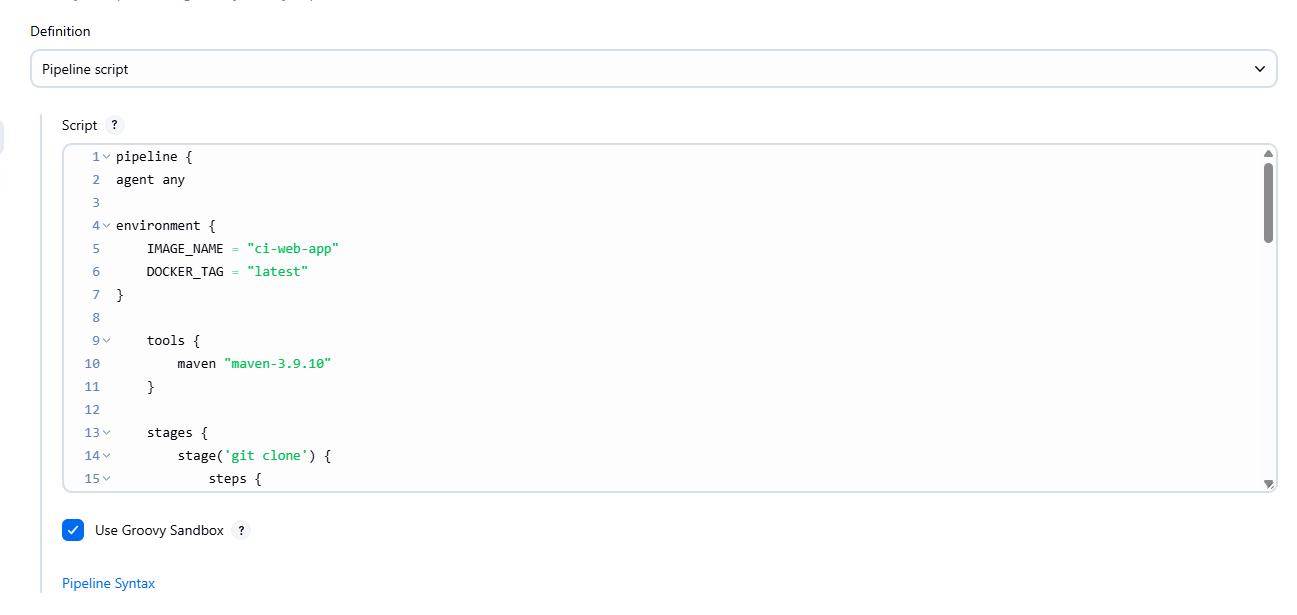
CD Pipeline was successful



Once the CI pipeline is successful, it needs to trigger CD pipeline automatically. How do we do it?

Modifying CI Pipeline Script

CI pipeline Pipeline Syntax



Build Build a Job

Select : CD\_WebApp\_Pipeline

Copy Script: build 'CD\_WebApp\_Pipeline'

pipeline {

agent any

environment {

IMAGE\_NAME = "ci-web-app"

DOCKER\_TAG = "latest"

}

tools {

maven "maven-3.9.10"

}

stages {

stage('git clone') {

steps {

git branch: 'main', url: 'https://github.com/Haider7214/WebAppMaven.git'

}

}

stage('maven build') {

steps {

sh 'mvn clean package'

}

}

stage('Build Docker Image') {

steps {

script {

writeFile file: 'Dockerfile', text: '''

# Use an official Tomcat base image

FROM tomcat:latest

LABEL maintainer="DemoDockerfile"

# Remove default webapps

RUN rm -rf /usr/local/tomcat/webapps/\*

# Copy WAR to Tomcat webapps

COPY target/\*.war /usr/local/tomcat/webapps/ROOT.war

# Expose port

EXPOSE 8080

'''

echo "✅ Dockerfile generated"

sh "docker build -t ${IMAGE\_NAME}:${DOCKER\_TAG} ."

}

}

}

stage('Docker push') {

steps {

withCredentials([string(credentialsId: 'Sai-Docker-Pwd', variable: 'Docker\_Hub\_PWD\_New')]) {

sh 'docker login -u saidocker567 -p ${Docker\_Hub\_PWD\_New}'

sh 'docker tag ${IMAGE\_NAME}:${DOCKER\_TAG} saidocker567/${IMAGE\_NAME}:${DOCKER\_TAG}'

sh 'docker push saidocker567/${IMAGE\_NAME}:${DOCKER\_TAG}'

}

}

}

stage('Trigger CD job if CI is successful') {

steps {

build 'CD\_WebApp\_Pipeline'

}

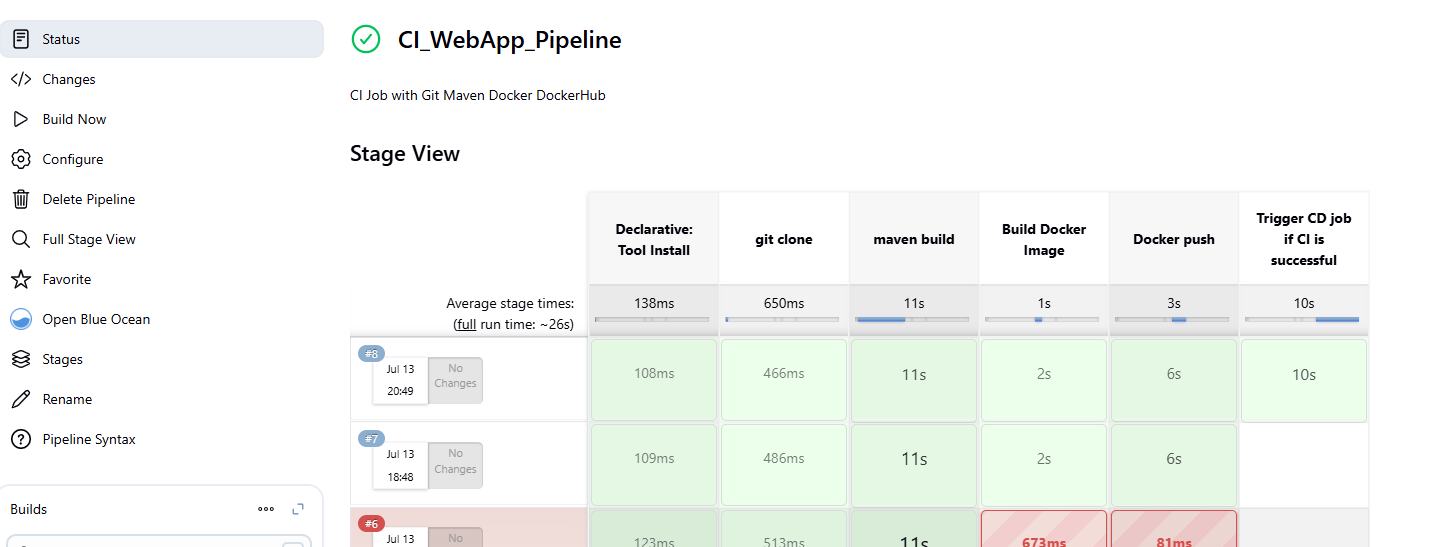
}

}

}

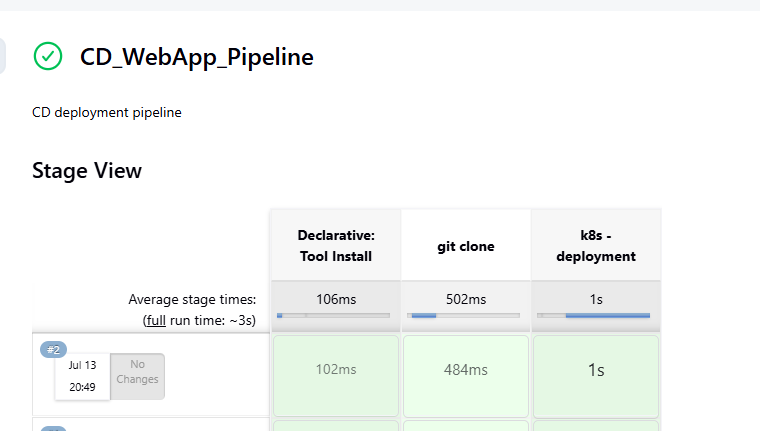
Apply and Save

Now it triggers CD job successfully



1:11

CD pipeline automatically getting executed



Go to Configure --> Poll SCM --> \*\*\*\*\* means every minute we have a new commit it will trigger automatically



Usually people use Terraform to provision the infrastructure

Jenkins will execute Terraform script to provision infrastructure

1:16

Run on EKS host

kubectl delete all --all

eksctl delete cluster --name my-eks-cluster --region ca-central-1

Make sure cluster is created on EKS-Host

Create EKS Cluster using eksctl

Create K8s cluster in **EKS-host VM**

eksctl create cluster --name my-eks-cluster --region ca-central-1 --node-type t2.medium --zones ca-central-1a,ca-central-1b

From EKS-host VM

ubuntu@ip-172-31-9-165:~$ cat .kube/config

apiVersion: v1

clusters:

- cluster:

certificate-authority-data: 

server: https://E359CB3780B483E3DEE92E09CDE915C8.gr7.ca-central-1.eks.amazonaws.com

name: my-eks-cluster.ca-central-1.eksctl.io

contexts:

- context:

cluster: my-eks-cluster.ca-central-1.eksctl.io

user: i-01289fc5ca918b25f@my-eks-cluster.ca-central-1.eksctl.io

name: i-01289fc5ca918b25f@my-eks-cluster.ca-central-1.eksctl.io

current-context: i-01289fc5ca918b25f@my-eks-cluster.ca-central-1.eksctl.io

kind: Config

preferences: {}

users:

- name: i-01289fc5ca918b25f@my-eks-cluster.ca-central-1.eksctl.io

user:

exec:

apiVersion: client.authentication.k8s.io/v1beta1

args:

- eks

- get-token

- --output

- json

- --cluster-name

- my-eks-cluster

- --region

- ca-central-1

command: aws

env:

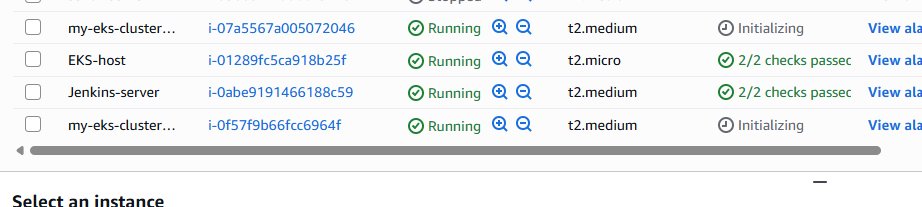
- name: AWS\_STS\_REGIONAL\_ENDPOINTS

value: regional

provideClusterInfo: false

Update in Jenkins Server

ubuntu@ip-172-31-11-116:~$ sudo vi .kube/config



ubuntu@ip-172-31-11-116:~$ sudo vi .kube/config

ubuntu@ip-172-31-11-116:~$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

ip-192-168-25-28.ca-central-1.compute.internal Ready <none> 25m v1.32.3-eks-473151a

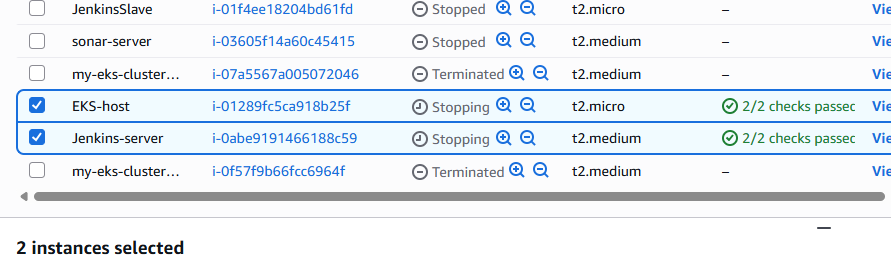
ip-192-168-61-42.ca-central-1.compute.internal Ready <none> 24m v1.32.3-eks-473151a

EKS-VM deleting cluster

kubectl delete all --all

eksctl delete cluster --name my-eks-cluster --region ca-central-1

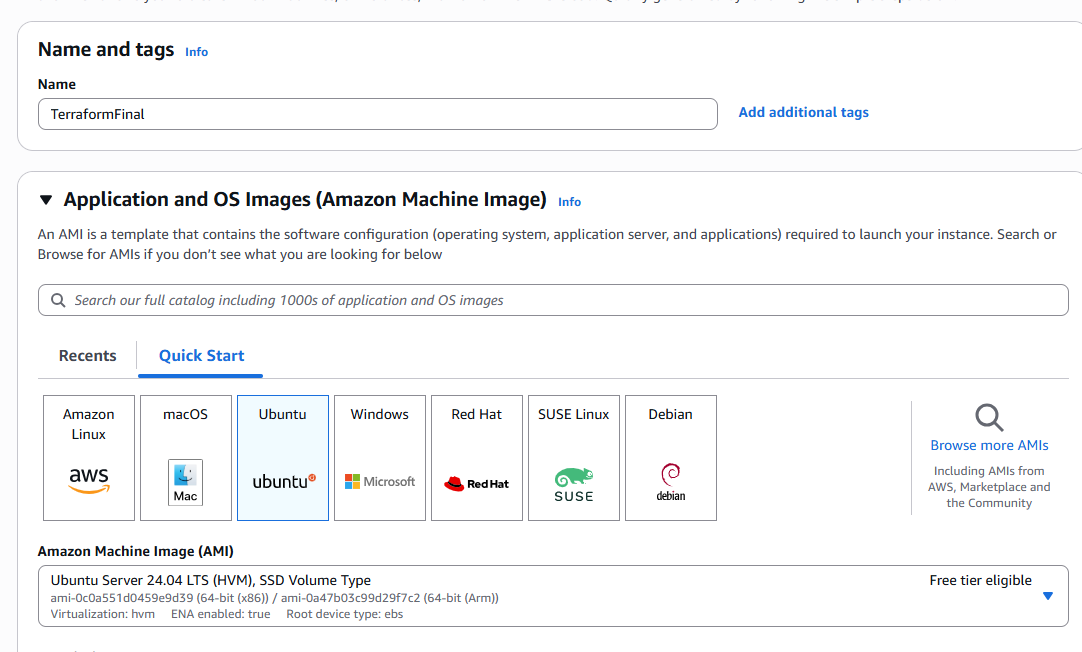
We don’t need these machines anymore so stopped them

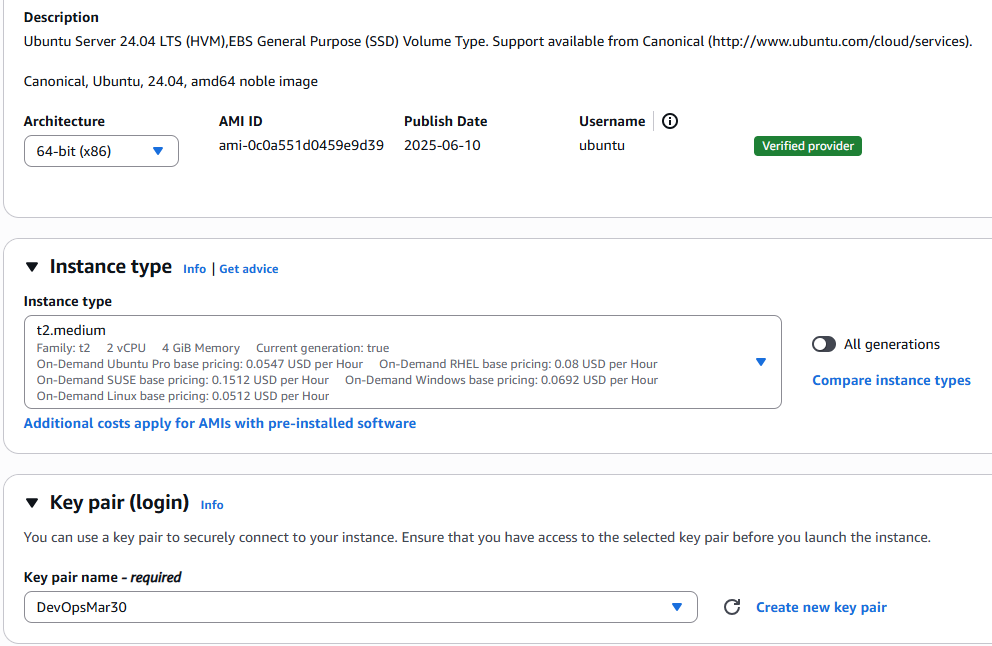


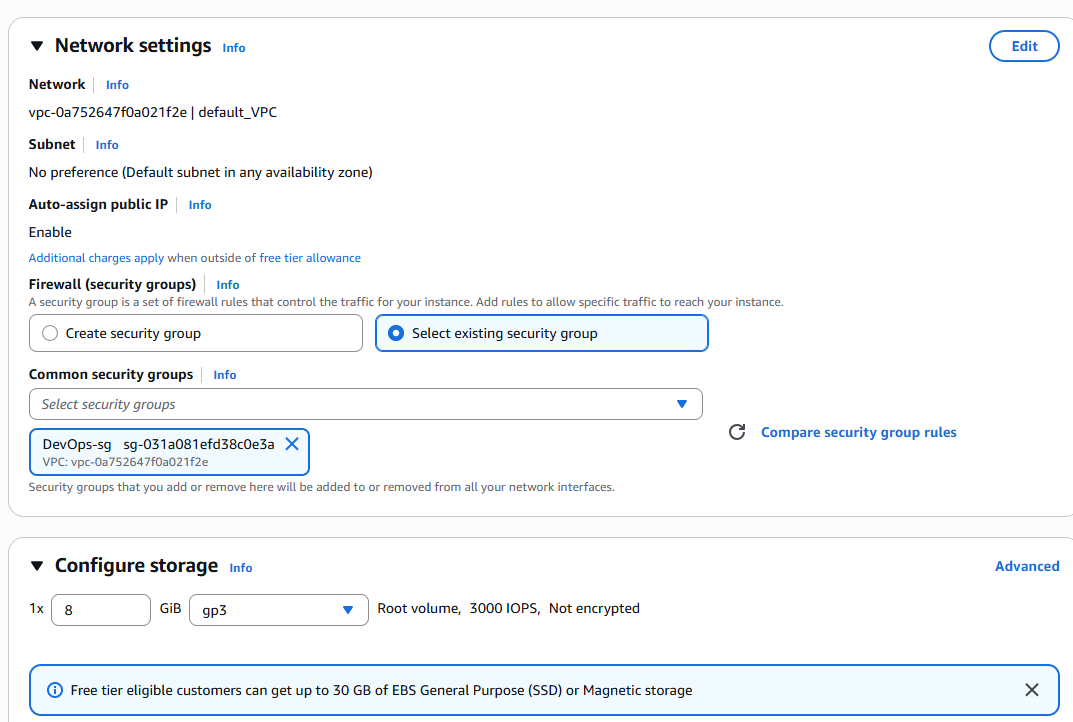
In the next Pipeline, we are going to automate Jenkins with Terraform to provision EKS cluster

Terraform --> Jenkins --> EKS cluster

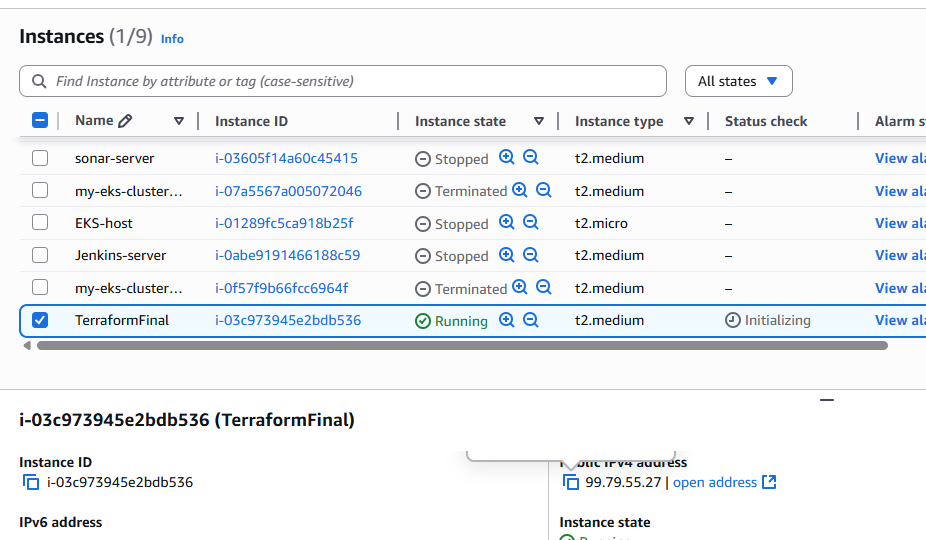
1. Have a machine to launch an EC2 instance (Ubuntu --> t2.medium) so we can have Terraform in it
2. Install Terraform, Java and Jenkins in this machine

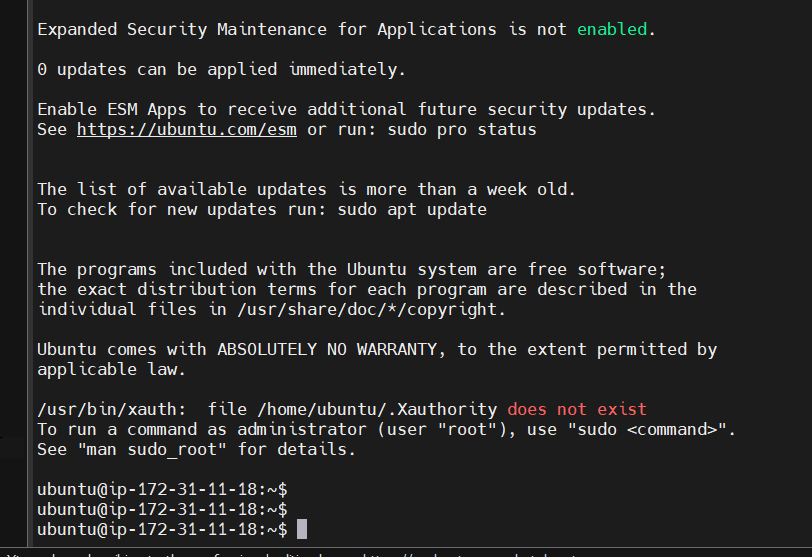






Launch Instance





1. Create Linux VM on AWS Cloud - Ubuntu ( preferred to use min t2.medium as instance type)

Get connected to Linux VM using ssh gitbash or terminal or any medium

2. install Java

1.sudo apt update -> update the package manager

2.sudo apt install openjdk-21-jdk -> install java

java -version -> To check java is installed or not

sudo apt update

sudo apt install openjdk-17-jdk

3. Install Jenkins

# Create keyring directory if it doesn't exist

sudo mkdir -p /etc/apt/keyrings

# Download and add the Jenkins GPG key

sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

# Add Jenkins repo to your sources list

echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/" \

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

sudo apt update

sudo apt install -y jenkins

4. Start and verify Jenkins

sudo systemctl enable jenkins

sudo systemctl start jenkins

Verify Jenkins

sudo systemctl status jenkins

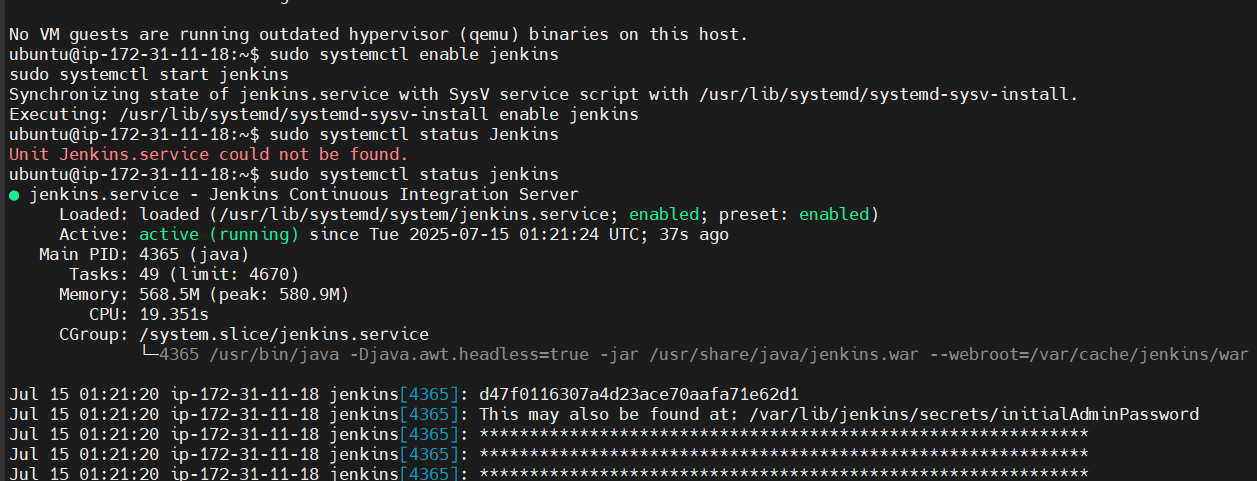
5. Open Jenkins server in browser ( also make sure edit inbond rules and add 8080 in security group)

http://public-ip:8080/

6: Copy Jenkins admin password

/var/lib/jenkins/secrets/initialAdminPassword

$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword



Install Terraform

ubuntu@ip-172-31-11-18:~$ sudo vi terraform.sh

#!/bin/bash

set -e

echo "🔄 Updating system packages..."

sudo apt-get update -y

sudo apt-get install -y curl unzip gnupg software-properties-common

echo "📥 Adding HashiCorp GPG key..."

curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "📦 Adding HashiCorp repo to apt sources..."

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" \

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

echo "🔄 Updating package list and installing Terraform..."

sudo apt-get update -y

sudo apt-get install terraform -y

echo "✅ Verifying Terraform version..."

terraform -version

echo "🎉 Terraform installation completed!"

ubuntu@ip-172-31-11-18:~$ sudo vi terraform.sh

ubuntu@ip-172-31-11-18:~$ ls

terraform.sh

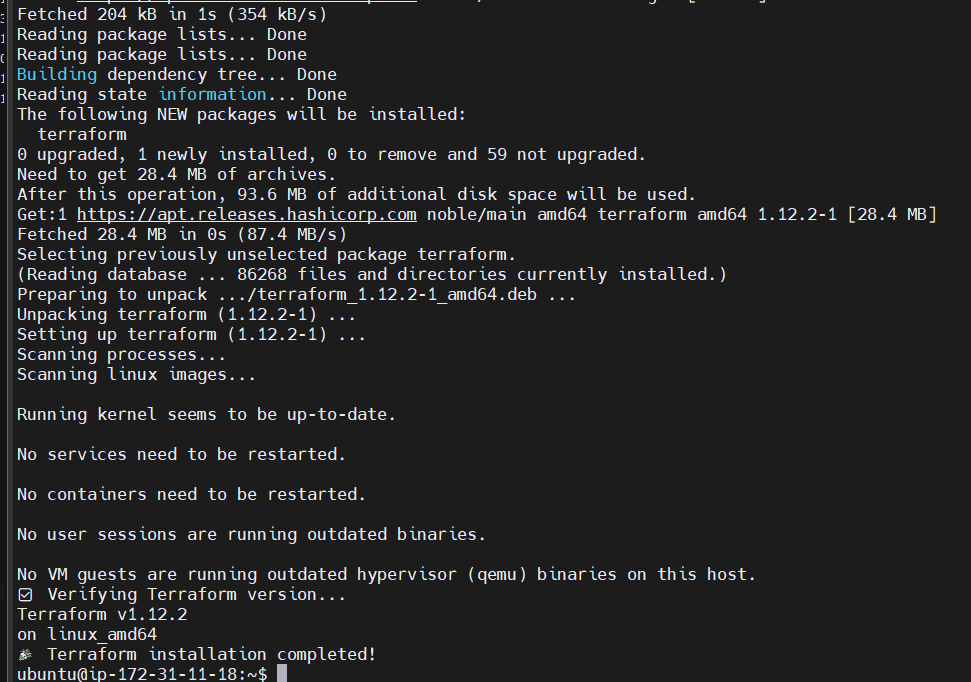
ubuntu@ip-172-31-11-18:~$ sudo chmod +x terraform.sh

ubuntu@ip-172-31-11-18:~$ ls -l

total 4

-rwxr-xr-x 1 root root 798 Jul 15 01:28 terraform.sh

ubuntu@ip-172-31-11-18:~$ sudo sh terraform.sh



1:40

ubuntu@ip-172-31-11-18:~$ sudo vi k8s.sh

#!/bin/bash

set -e

echo "🔄 Updating system packages..."

sudo apt-get update -y

echo "📥 Downloading latest kubectl binary..."

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

echo "🔐 Verifying the binary checksum..."

curl -LO "https://dl.k8s.io/release/$(curl -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"

echo "$(cat kubectl.sha256) kubectl" | sha256sum --check

echo "📦 Installing kubectl..."

chmod +x kubectl

sudo mv kubectl /usr/local/bin/

echo "✅ Verifying kubectl version..."

kubectl version --client

echo "🎉 kubectl installed successfully!"

ubuntu@ip-172-31-11-18:~$ sudo vi k8s.sh

ubuntu@ip-172-31-11-18:~$ sudo chmod +x k8s.sh

ubuntu@ip-172-31-11-18:~$ sh k8s.sh

#!/bin/bash

set -e

echo "🔄 Updating system packages..."

sudo apt-get update -y

echo "📥 Fetching latest stable version..."

KUBECTL\_VERSION=$(curl -L -s https://dl.k8s.io/release/stable.txt)

echo "📥 Downloading kubectl version $KUBECTL\_VERSION..."

curl -LO "https://dl.k8s.io/release/${KUBECTL\_VERSION}/bin/linux/amd64/kubectl"

echo "🔐 Downloading checksum..."

curl -LO "https://dl.k8s.io/release/${KUBECTL\_VERSION}/bin/linux/amd64/kubectl.sha256"

echo "✅ Verifying checksum..."

echo "$(cat kubectl.sha256) kubectl" | sha256sum --check -

echo "🚀 Installing kubectl..."

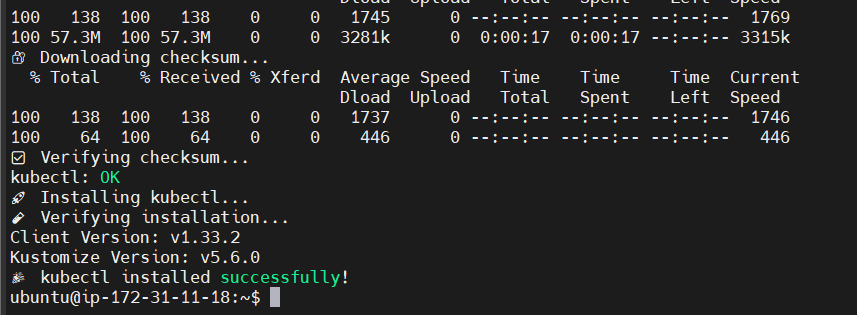
chmod +x kubectl

sudo mv kubectl /usr/local/bin/

echo "🧪 Verifying installation..."

kubectl version --client

echo "🎉 kubectl installed successfully!"



ubuntu@ip-172-31-11-18:~$ sudo vi aws-cli.sh

ubuntu@ip-172-31-11-18:~$ sudo chmod +x aws-cli.sh

#!/bin/bash

set -e

echo "🔄 Updating system packages..."

sudo apt-get update -y

sudo apt-get install -y unzip curl

echo "📥 Downloading AWS CLI v2..."

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

echo "📦 Unzipping..."

unzip -q awscliv2.zip

echo "🚀 Installing AWS CLI..."

sudo ./aws/install

echo "🧹 Cleaning up..."

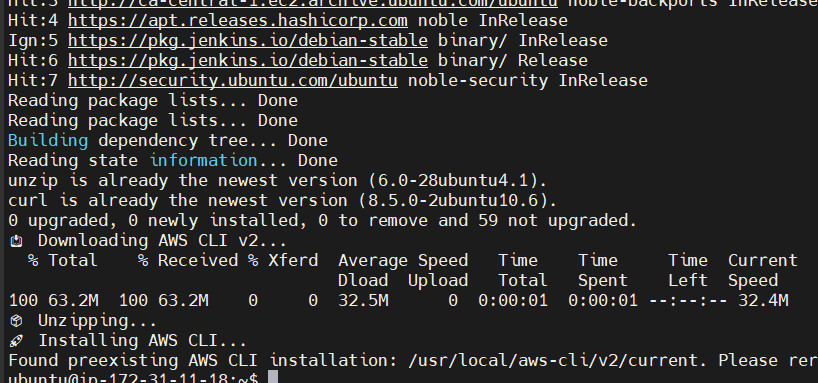
rm -rf aws awscliv2.zip

echo "✅ Verifying AWS CLI installation..."

aws --version

echo "🎉 AWS CLI installed successfully!"

ubuntu@ip-172-31-11-18:~$ sh aws-cli.sh



1:46

install kubectl

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

# Make executable and move to /usr/local/bin

chmod +x kubectl

sudo mv kubectl /usr/local/bin/

# Verify installation

kubectl version --client --output=yaml

install AWS CLI

# Install unzip (adjust for your package manager)

# For Debian/Ubuntu

$ sudo apt update && sudo apt install -y unzip

# For RHEL/CentOS

# sudo yum install -y unzip

# Download and install AWS CLI v2

$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

# Clean up

$ rm -rf awscliv2.zip aws

# Verify installation

$ aws --version

ubuntu@ip-172-31-11-18:~$ sudo chmod +x eksctl.sh

Install eksctl

# Download and extract the latest eksctl

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp

# Verify installation

eksctl version

#!/bin/bash

set -e

echo "🔄 Updating system packages..."

sudo apt-get update -y

sudo apt-get install -y curl tar

echo "📥 Fetching latest eksctl version..."

LATEST\_VERSION=$(curl -s https://api.github.com/repos/eksctl-io/eksctl/releases/latest | grep tag\_name | cut -d '"' -f 4)

if [[ -z "$LATEST\_VERSION" ]]; then

echo "❌ Failed to retrieve eksctl version. Check your network or GitHub rate limits."

exit 1

fi

echo "➡️ Latest version is $LATEST\_VERSION"

TAR\_NAME="eksctl\_Linux\_amd64.tar.gz"

DOWNLOAD\_URL="https://github.com/eksctl-io/eksctl/releases/download/${LATEST\_VERSION}/${TAR\_NAME}"

echo "📥 Downloading from: $DOWNLOAD\_URL"

curl -LO "$DOWNLOAD\_URL"

echo "📦 Extracting..."

tar -xzf "$TAR\_NAME"

echo "🚀 Installing eksctl to /usr/local/bin..."

sudo mv eksctl /usr/local/bin/

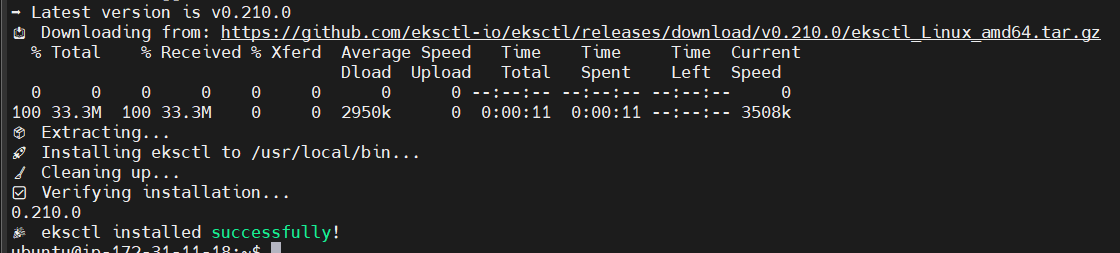
echo "🧹 Cleaning up..."

rm -f "$TAR\_NAME"

echo "✅ Verifying installation..."

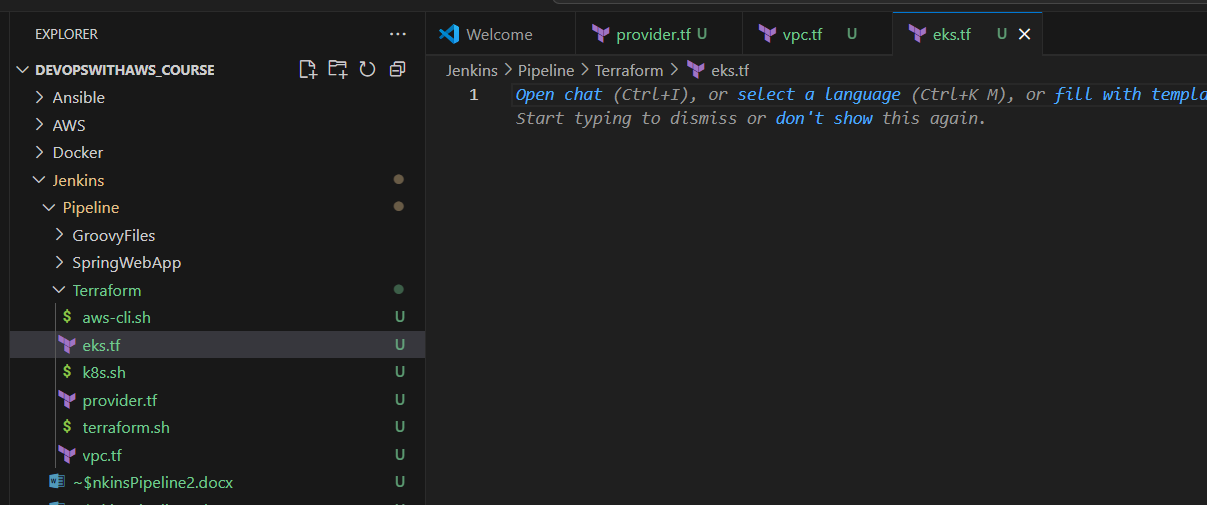
eksctl version

echo "🎉 eksctl installed successfully!"



1:50

Go to VSCode create 3 tf files



1:51

To access resources over internet, we use Public subnets

provider.tf

locals {

region = "ap-south-1"

name = "telusko-eks-cluster"

vpc\_cidr = "10.123.0.0/16"

azs = ["ap-south-1a", "ap-south-1b"]

public\_subnets = ["10.123.1.0/24", "10.123.2.0/24"]

private\_subnets = ["10.123.3.0/24", "10.123.4.0/24"]

intra\_subnets = ["10.123.5.0/24", "10.123.6.0/24"]

tags = {

Example = local.name

}

}

provider "aws" {

region = "ap-south-1"

}

vpc.tf

module "vpc {

source = "terraform-aws-modules/vpc/aws"

version = "~> 4.0" # Latest 4.x version of VPC to be used

name = local.name

cidr = local.vpc\_cidr

azs = local.azs

private\_subnets = local.private\_subnets

public\_subnets = local.public\_subnets

intra.intra\_subnets = local.intra\_subnets

enable\_nat\_gateway = true # if Private subnets require internet access to access external APIs or resources then we need to enable the NAT Gateway

public\_subnets\_tags = {

"kubernetes.io/role/elb" = 1

}

private\_subnets\_tags = {

"kubernetes.io/role/internal-elb" = 1

}

}

eks.tf

module "eks" {

source = "terraform-aws-modules/eks/aws" # specifies location of module from Terraform AWS registry

version = "19.15.1"

cluster\_name = local.name

cluster\_endpoint\_public\_access = true # to enable public access to all cluster endpoints

cluster\_addons = {

coredns = {

most\_recent = true

}

kube-proxy = { # for kubernetes API communication within the cluster for networking purpose, kube-proxy is required

most\_recent = true

}

vpc-cni = {

most\_recent = true

}

}

# for networking

vpc\_id = module.vpc.vpc\_id

subnet\_ids = module.vpc.private\_subnets

control\_plane\_subnet\_ids = module.vpc.intra\_subnets

# what type of machines you want

eks\_managed\_node\_group\_defaults = {

ami\_type = "AL2\_x86\_64"

instance\_types = ["t2.medium"]

attach\_cluster\_primary\_security\_group = true

}

eks\_managed\_node\_groups = {

pipeline-cluster-wg = {

min\_size = 2

max\_size = 2

desired\_size = 2

instance\_types = ["t2.medium"]

capacity\_type = "SPOT"

tags = {

ExtraTag = "full\_pipeline"

}

}

}

tags = local.tags

}

Next class building the pipeline