Jenkins Pipeline 2

CICD --> Jenkins Pipeline

Final project --> Terraform to automate React + SpringBoot app

1. GitHub
2. Maven in Jenkins pipeline stage
3. Docker image we push into Docker registry (stage)
4. Deploy application using Kubernetes cluster
5. Setup pipeline using Jenkins

Docker to create image stage 4

Jenkins pipeline stage 2

GitHub stage 1

Maven to build that project stage 3

Push image into DockerHub stage 4

Deploy using K8s cluster stage 5

GitHub repo --> Git clone (Stage 1) --> Build Jar/War (Stage 2) --> Docker build the image (Stage 3) --> Docker hub and push image (Stage 4) --> K8s to deploy application (Stage 5) . We will have Manifest.yml file then we will access application from browser

Final CICD pipelines

Jenkins pipeline 1 --> Git (Project clone) + Maven (Project build) + Docker (Docker image creation) + Docker Hub (Pushing image into Docker registry) + Kubernetes (Deploy application)

Until Pushing image into Docker registry it is CI and Kubernetes part is CD

============================================

1 ==> Create EKS Host VM is Created in AWS

=============================================

a) Launch a Linux machine (Ubuntu VM ) using AWS EC2 (t2.micro)

b) Connect to this machine and install kubectl

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

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

=========================================================

2 ==> Create IAM Role and attach to EKS Management HOST

=========================================================

a)Create a new Role using IAM Service (Select Use case EC2)

b)Add below permission

AdministratorAccess

AmazonEC2FullAccess

AmazonVPCFullAccess

AWSCloudFormationFullAccess

IAMFullAccess

c)Enter Role Name(telusko\_eks\_role)

d) Attach created role to EKS management host vm --> Actions-> Security-> Modify IAM role --> add created IAM role

=============================================

3 ==> Create EKS Cluster using eksctl

=============================================

eksctl create cluster --name telusko-cluster --region ap-south-1 --node-type t2.medium --zones ap-south-1a,ap-south-1b

kubeconfig as "/home/ubuntu/.kube/config"

EKS cluster "telusko-cluster" in "ap-south-1" region is ready

$ cat /home/ubuntu/.kube/config

$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

ip-192-168-27-229.ap-south-1.compute.internal Ready <none> 12m v1.32.3-eks-473151a

ip-192-168-39-106.ap-south-1.compute.internal Ready <none> 12m v1.32.3-eks-473151a

Also check on AWS Console that cluster and also two new instance worker nodes would be created

=======================================================

4 ==> Jenkins Server Setup in Linux VM

=======================================================

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

===========================================================================================================

===> Attach created role to Jenkins Server host vm --> Actions-> Security-> Modify IAM role --> add created iam role

=============================================================================================================

====================================================

5 ==> Configure Maven as Global Tool in Jenkins

====================================================

Manage Jenkins --> Tools --> Maven Installation --> Add maven

==========================================

6 ==> Docker Setup in Jenkins

==========================================

--> Execute the commands

sudo apt update

sudo apt install -y ca-certificates curl gnupg lsb-release

echo \

"deb [arch=$(dpkg --print-architecture) \

signed-by=/etc/apt/keyrings/docker.gpg] \

https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable" | \

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

echo \

"deb [arch=$(dpkg --print-architecture) \

signed-by=/etc/apt/keyrings/docker.gpg] \

https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable" | \

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

sudo apt update

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

sudo systemctl enable docker

sudo systemctl start docker

Add jenkins into docker group

$ sudo usermod -aG docker jenkins

$ sudo systemctl restart jenkins

$ sudo docker -v ---> Verify Docker installation

$ sudo docker version

===================================================

7 ==> Install AWS CLI and Kubectl in Jenkins Server

===================================================

install aws cli

sudo apt install unzip

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

unzip awscliv2.zip

sudo ./aws/install

aws --version

--------------------------------------

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

=========================================================

8 ==> Update EKS Cluster Config File in Jenkins Server

=========================================================

Execute the below command in EKS Management Machine and Copy config file data

$ cat .kube/config

Connect to Jenkins server execute the following command to add config file into Jenkins server

$ cd /var/lib/Jenkins

$ sudo mkdir .kube

$ sudo vi .kube/config

( paste config file data copied from eks host machine )

Check eks nodes

$ kubectl get nodes

$ cd ~

$ ls -la

$ sudo vi .kube/config

( paste config file data copied from eks host machine )

$ kubectl get nodes

================================================================

9 ==> Create Jenkins CICD Pipeline with all Stages

================================================================

stage - 1 ==> Clone Git Repo

stage - 2 ==> Maven Build

stage - 3 ==> Create Docker Image

stage - 4 ==> Push Docker Image to Repository

stage - 5 ==> Deploy app in K8s eks Cluster

=====================================

10 ==> Install Blue Ocean Plugin

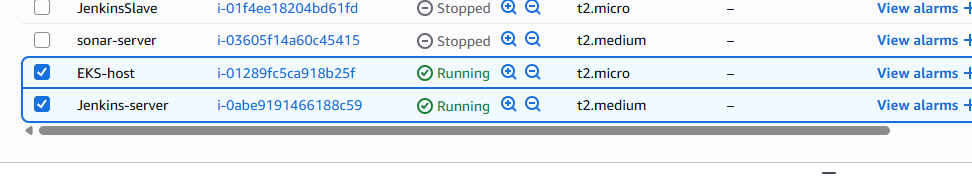
=====================================

Manage Jenkins --> plugins --> Available plugins --> Search Blue ocean --> install

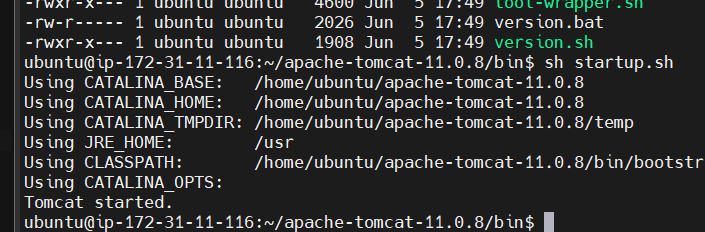
You will see that in Jenkins dashboard and use the same to work with your pipeline

==========================================================

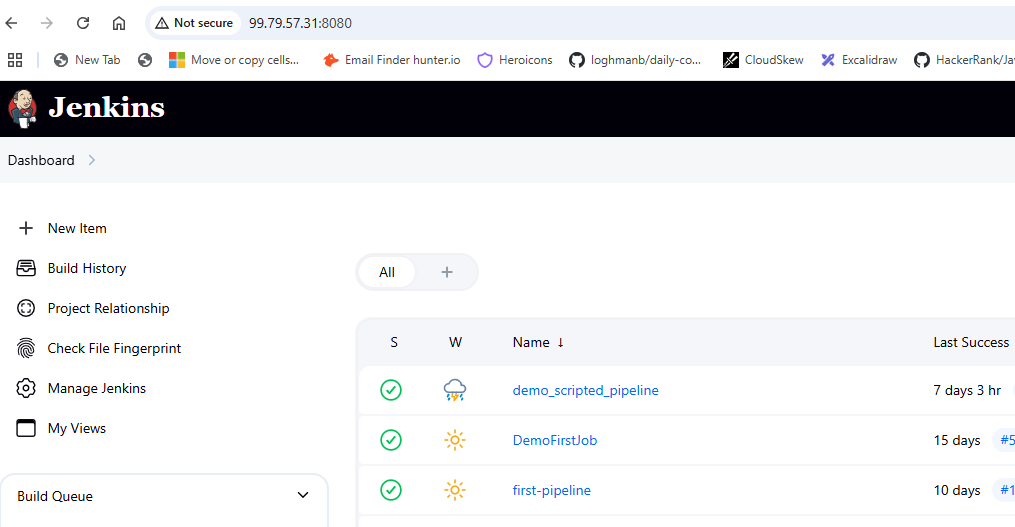
Jenkins Pipeline Step by Step

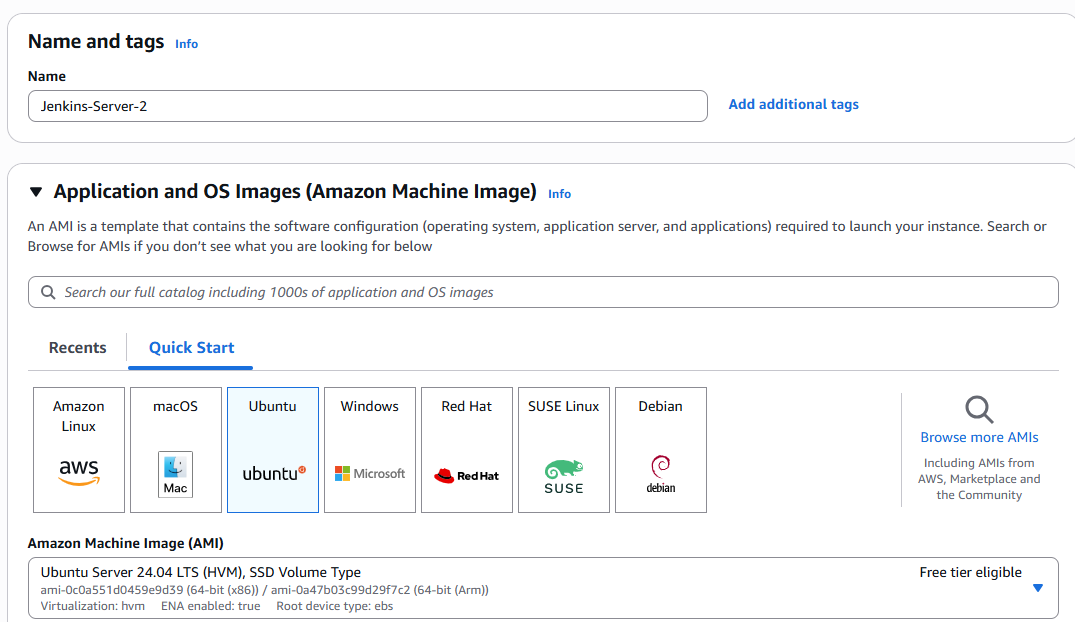


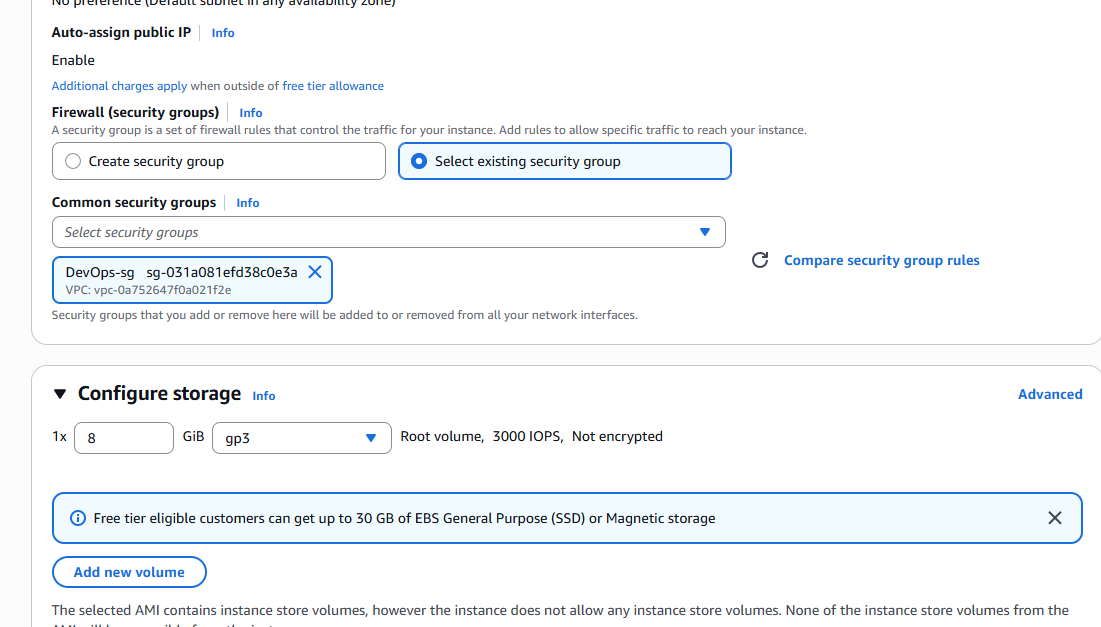
Log into Jenkins-server



Log into Jenkins

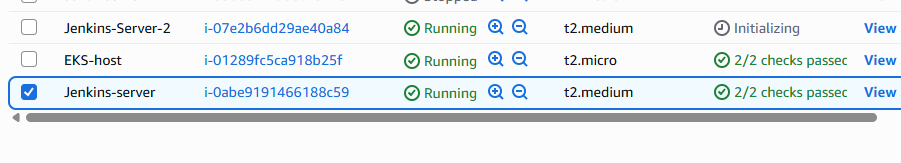






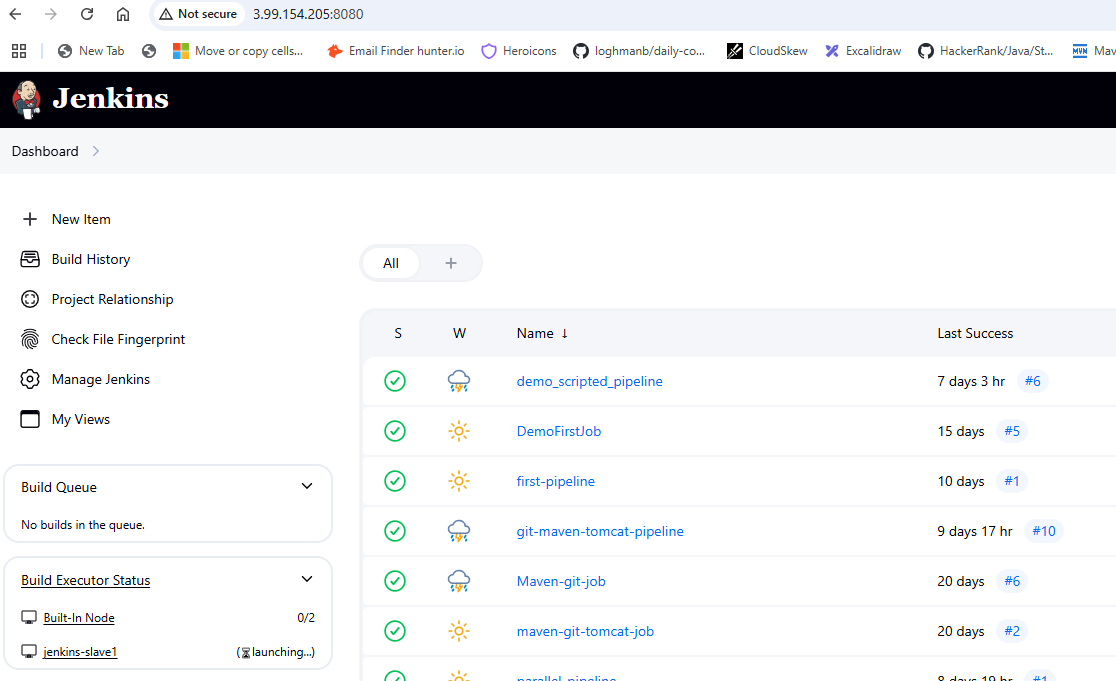
Launch Instance

Lets use Jenkins-Server-2

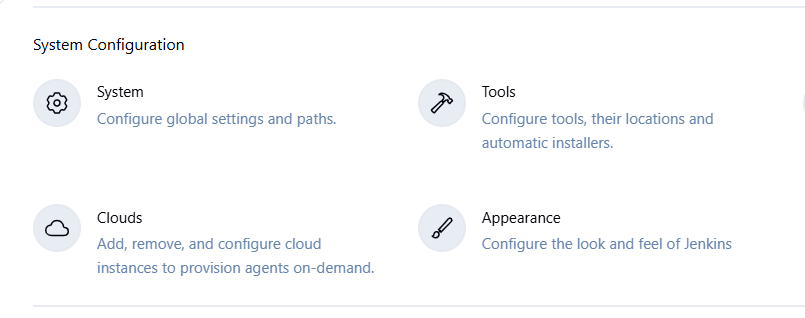


I am trying with Jenkins-Server original Jenkins server first

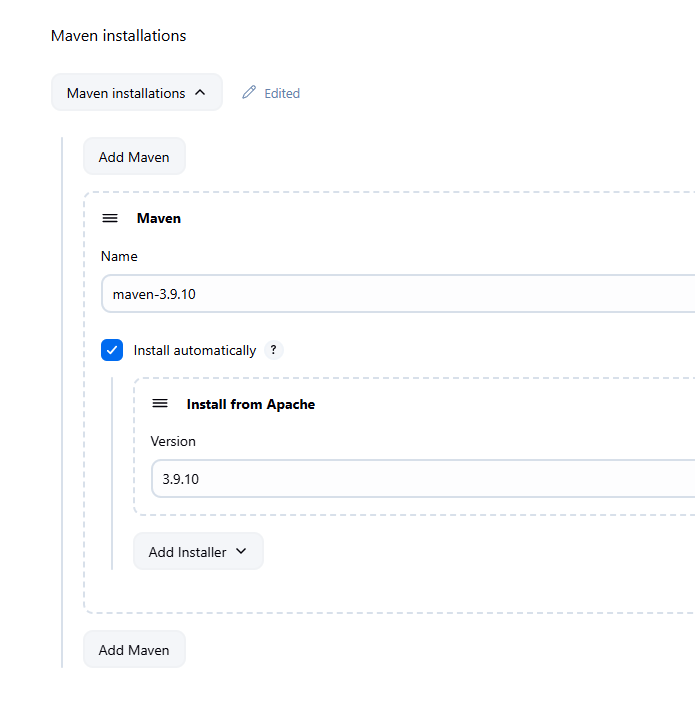
Logged back in



Go to Tools



Maven is already there



Docker Setup in Jenkins

# 1. Update existing packages

sudo apt update

sudo apt upgrade -y

# 2. Install dependencies

sudo apt install -y \

ca-certificates \

curl \

gnupg \

lsb-release

# 3. Add Docker’s official GPG key

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \

sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

# 4. Set up the Docker repo

echo \

"deb [arch=$(dpkg --print-architecture) \

signed-by=/etc/apt/keyrings/docker.gpg] \

https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) stable" | \

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

# 5. Update package index with Docker repo

sudo apt update

# 6. Install Docker packages

sudo apt install -y \

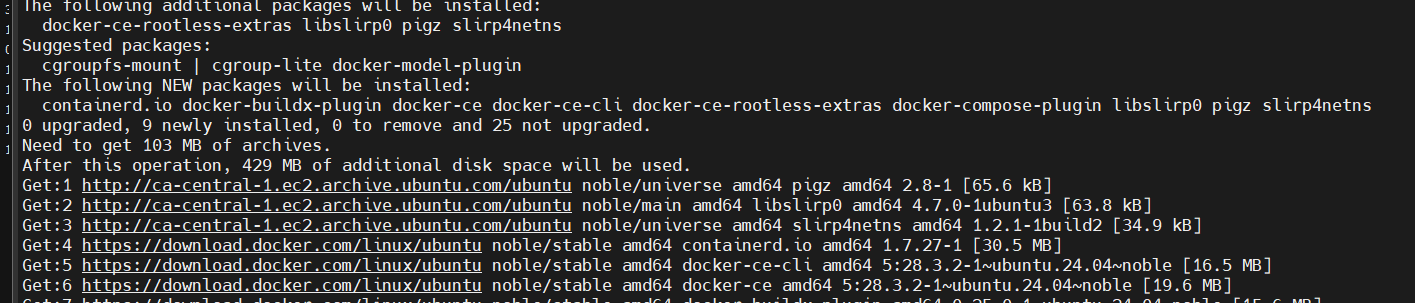
docker-ce \

docker-ce-cli \

containerd.io \

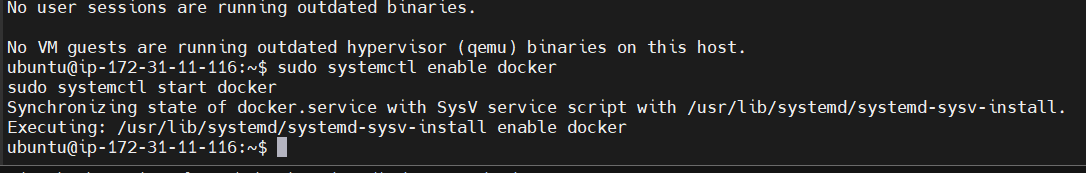
docker-buildx-plugin \

docker-compose-plugin



sudo systemctl enable docker

sudo systemctl start docker



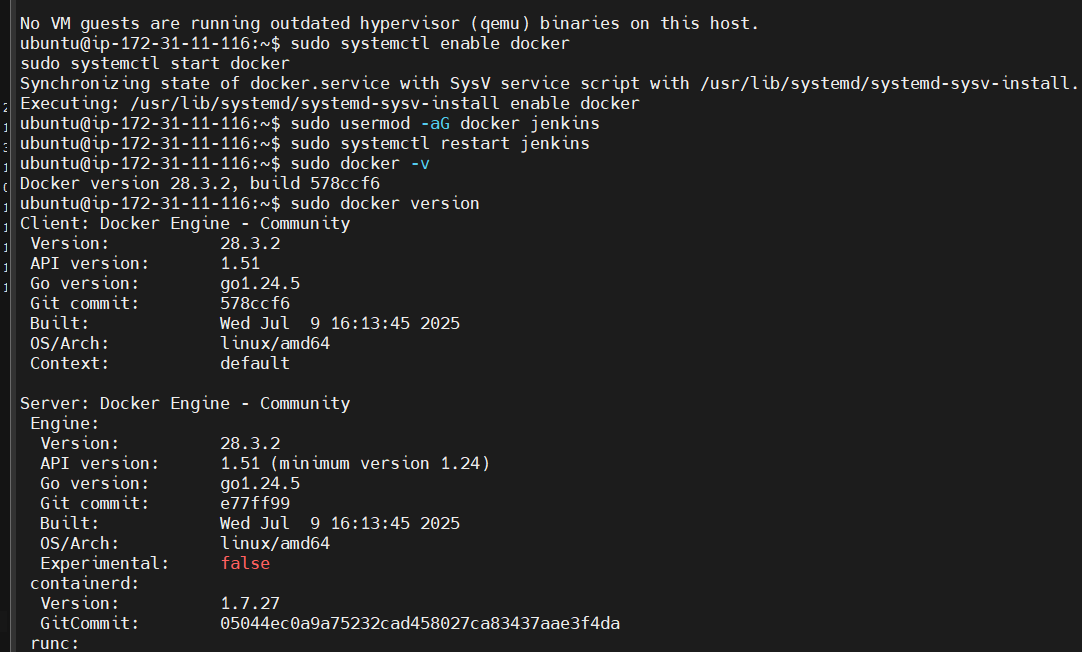
Add jenkins into docker group

$ sudo usermod -aG docker jenkins

$ sudo systemctl restart jenkins

$ sudo docker -v ---> Verify Docker installation

$ sudo docker version



Install AWS CLI and Kubectl in Jenkins Server

install aws cli

sudo apt install unzip

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

unzip awscliv2.zip

sudo ./aws/install

aws --version

--------------------------------------

install kubectl

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

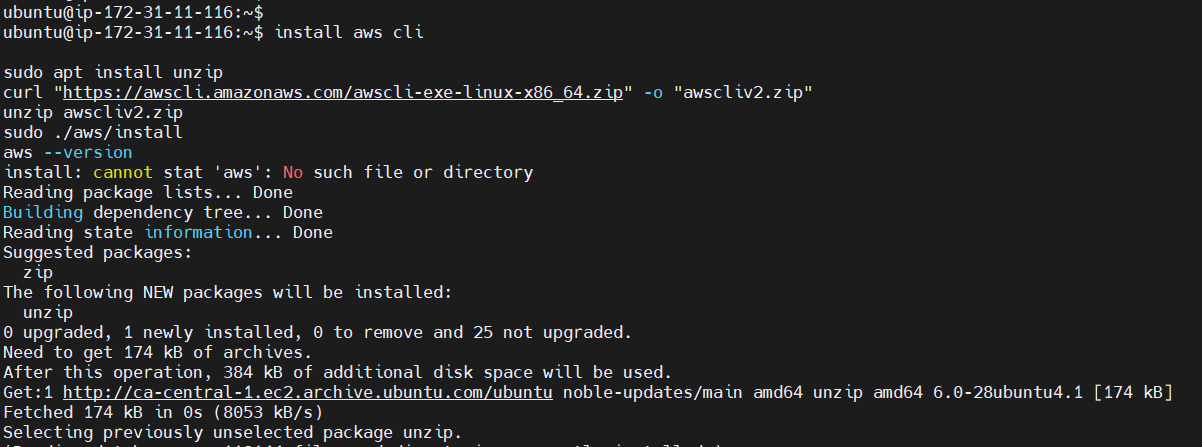
# Make executable and move to /usr/local/bin

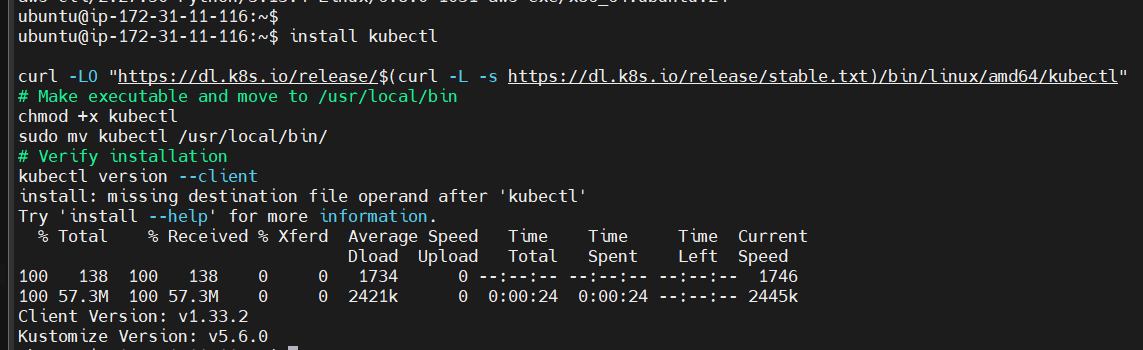
chmod +x kubectl

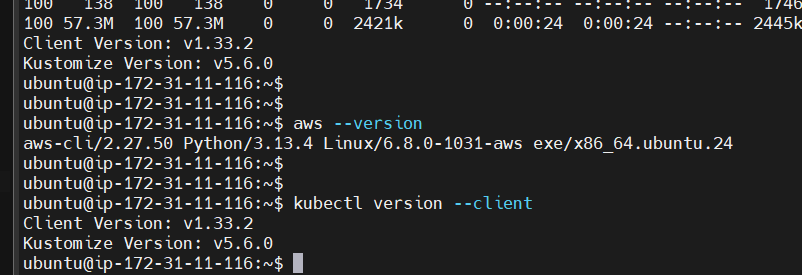
sudo mv kubectl /usr/local/bin/

# Verify installation

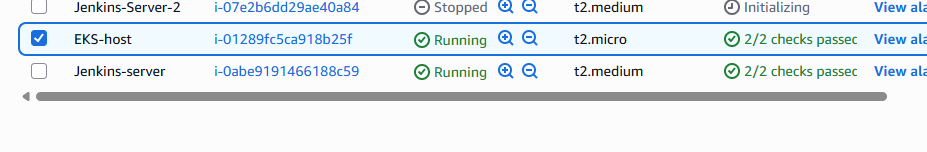
kubectl version --client

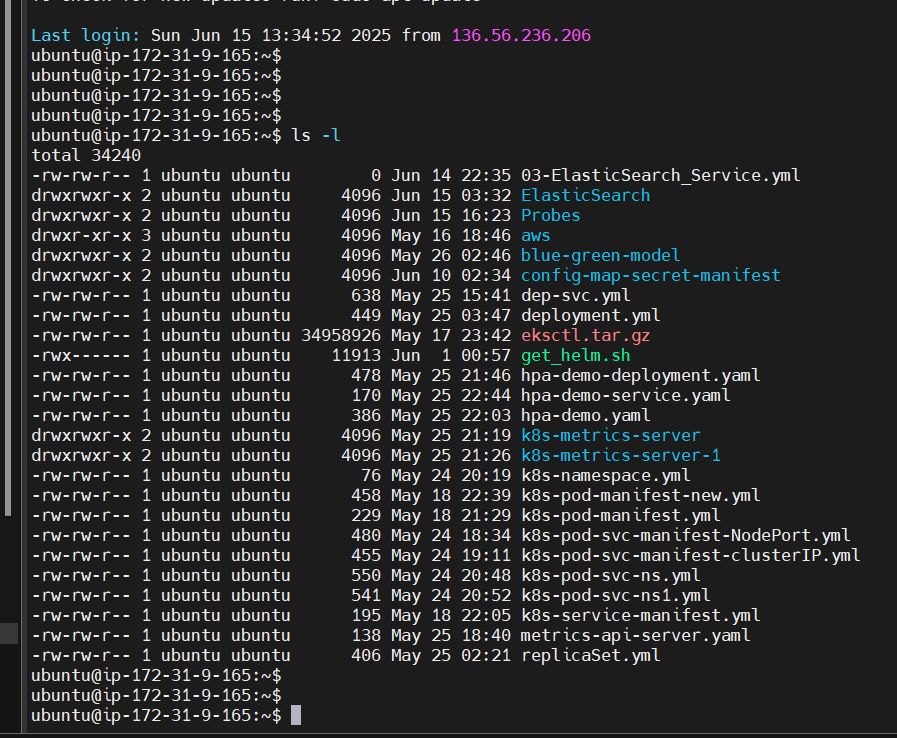






Start EKS-host machine

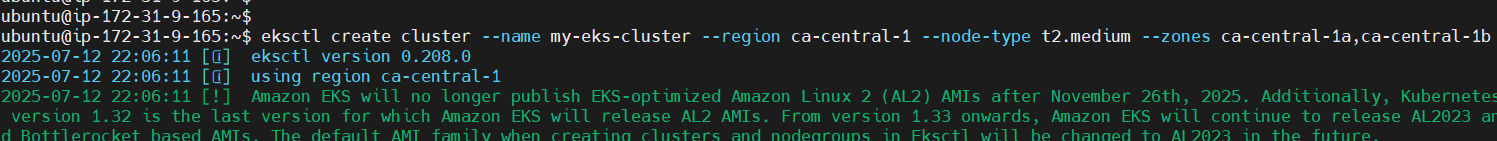




Create EKS Cluster using eksctl

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



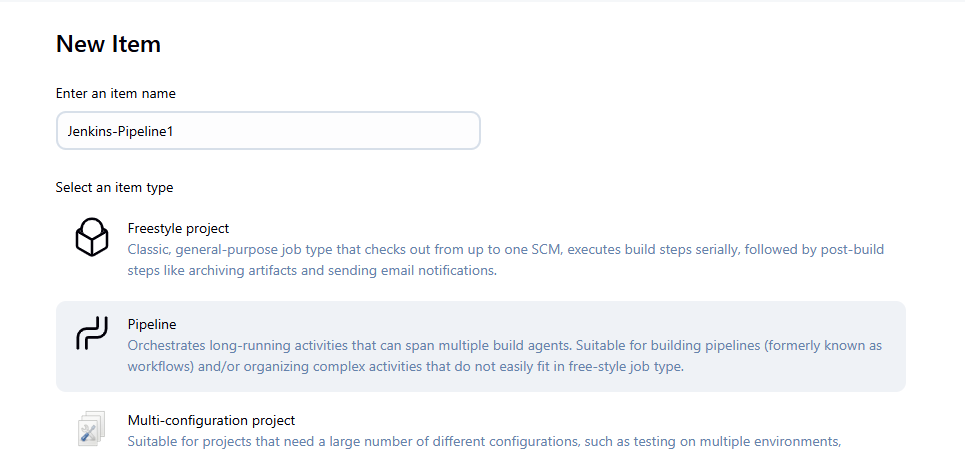
kubeconfig as "/home/ubuntu/.kube/config"

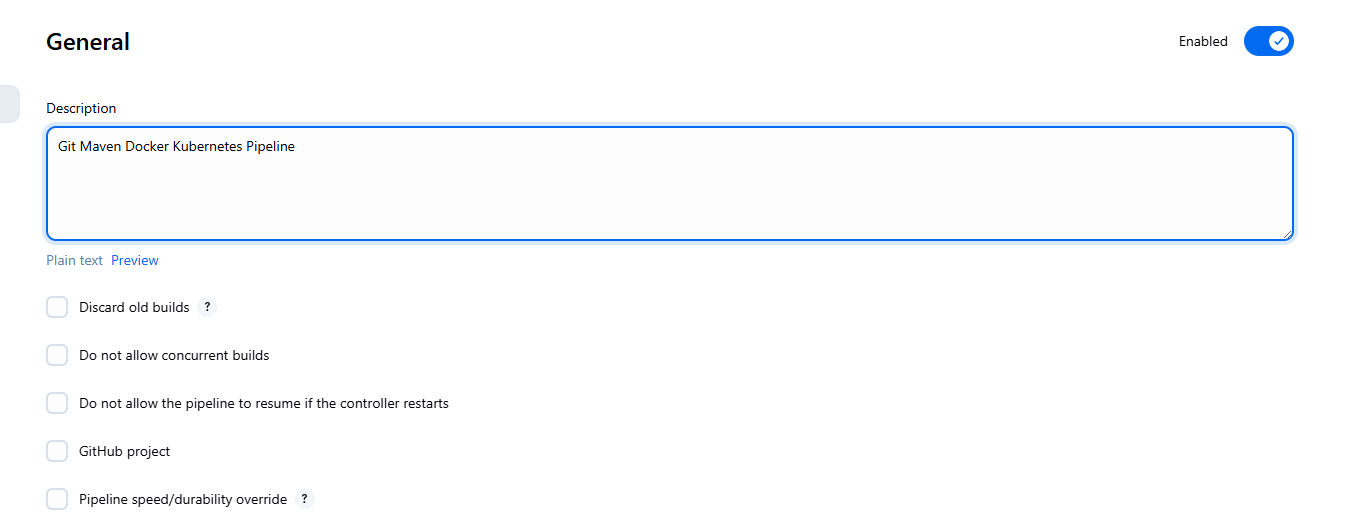
EKS cluster "telusko-cluster" in "ap-south-1" region is ready

$ cat /home/ubuntu/.kube/config

$ kubectl get nodes

New Item on Jenkins Pipeline





@Library('demo\_shared\_lib')\_

pipeline {

agent any

tools {

maven "maven-3.9.10"

}

stages {

stage('git clone') {

steps {

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

}

}

stage('demo message') {

steps {

telusko()

}

}

stage('maven build') {

steps {

mavenBuild()

}

}

stage('parallel stage') {

parallel {

stage('code-review'){

steps {

echo 'code review'

}

}

stage('nexus-upload'){

steps {

echo 'nexus upload'

}

}

}

}

stage('app deployed') {

steps {

echo 'Deploying App with Tomcat'

}

}

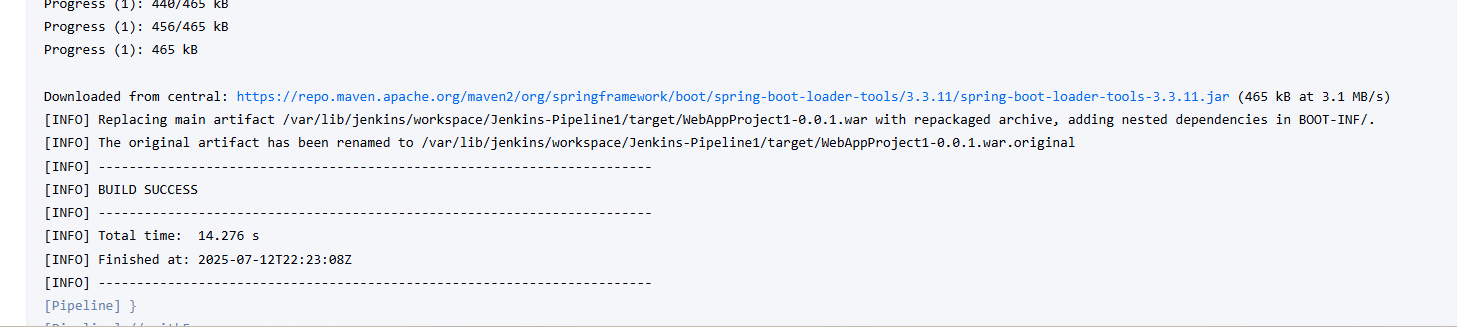
}

}

Apply and Save

Build Now

Build was successful



@Library('demo\_shared\_lib')\_

pipeline {

agent any

tools {

maven "maven-3.9.10"

}

stages {

stage('git clone') {

steps {

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

}

}

stage('demo message') {

steps {

telusko()

}

}

stage('maven build') {

steps {

mavenBuild()

}

}

stage('Find WAR file') {

steps {

script {

echo "Searching for WAR file..."

def warPath = sh(

script: "find target -name '\*.war' | head -n 1",

returnStdout: true

).trim()

if (warPath) {

echo "✅ WAR file found: ${warPath}"

} else {

error("❌ WAR file not found!")

}

}

}

}

stage('parallel stage') {

parallel {

stage('code-review'){

steps {

echo 'code review'

}

}

stage('nexus-upload'){

steps {

echo 'nexus upload'

}

}

}

}

stage('app deployed') {

steps {

echo 'Deploying App with Tomcat'

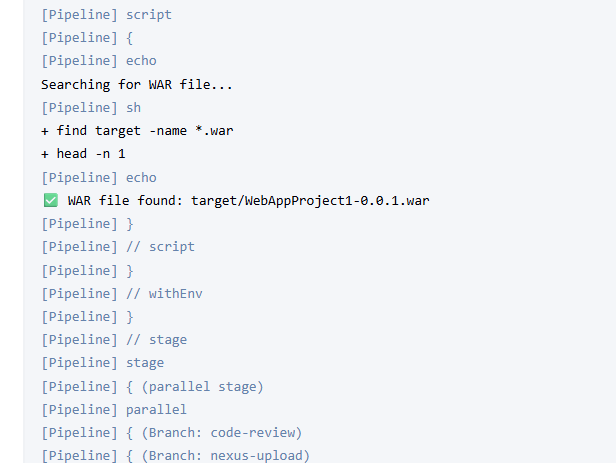
}

}

}

}

Build was successful



WAR file found: target/WebAppProject1-0.0.1.war

Create Dockerfile

@Library('demo\_shared\_lib')\_

pipeline {

agent any

environment {

IMAGE\_NAME = “my-web-app”

DOCKER\_TAG=”latest”

}

tools {

maven "maven-3.9.10"

}

stages {

stage('git clone') {

steps {

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

}

}

stage('demo message') {

steps {

telusko()

}

}

stage('maven build') {

steps {

sh ‘mvn clean compile test package’

}

}

stage('Find WAR file') {

steps {

script {

echo "Searching for WAR file..."

def warPath = sh(

script: "find target -name '\*.war' | head -n 1",

returnStdout: true

).trim()

if (warPath) {

echo "✅ WAR file found: ${warPath}"

} else {

error("❌ WAR file not found!")

}

}

}

}

stage('Build Docker Image') {

steps {

script {

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

# Use an official Tomcat base image

FROM tomcat:11-jdk21

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/

# Expose port

EXPOSE 8080

'''

echo "✅ Dockerfile generated"

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

}

}

}

}

@Library('demo\_shared\_lib')\_

pipeline {

agent any

environment {

IMAGE\_NAME = "my-web-app"

DOCKER\_TAG = "latest"

}

tools {

maven "maven-3.9.10"

}

stages {

stage('git clone') {

steps {

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

}

}

stage('demo message') {

steps {

telusko()

}

}

stage('maven build') {

steps {

sh 'mvn clean compile test package'

}

}

stage('Find WAR file') {

steps {

script {

echo "Searching for WAR file..."

def warPath = sh(

script: "find target -name '\*.war' | head -n 1",

returnStdout: true

).trim()

if (warPath) {

echo "✅ WAR file found: ${warPath}"

} else {

error("❌ WAR file not found!")

}

}

}

}

stage('Build Docker Image') {

steps {

script {

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

# Use an official Tomcat base image

FROM tomcat:11-jdk21

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/

# Expose port

EXPOSE 8080

'''

echo "✅ Dockerfile generated"

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

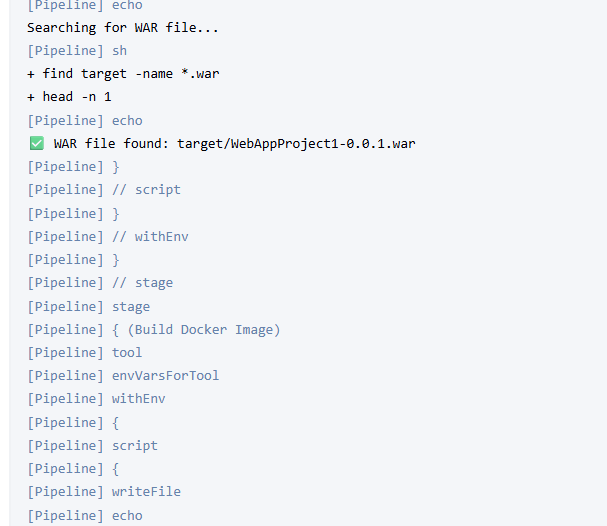
}

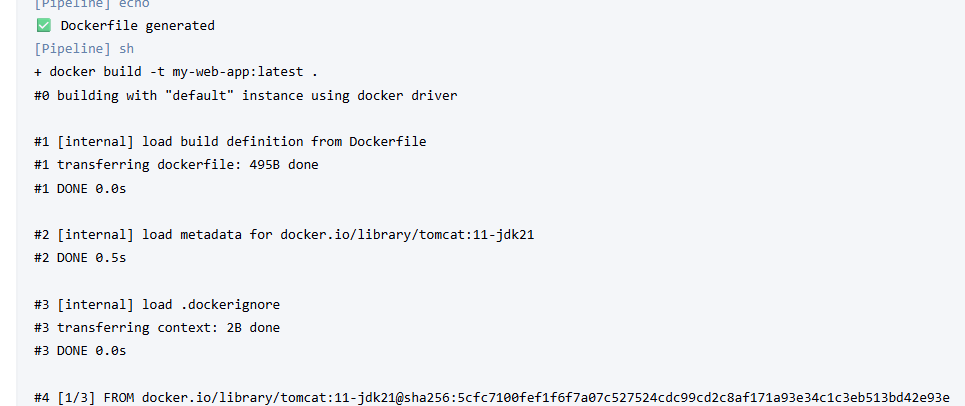
}

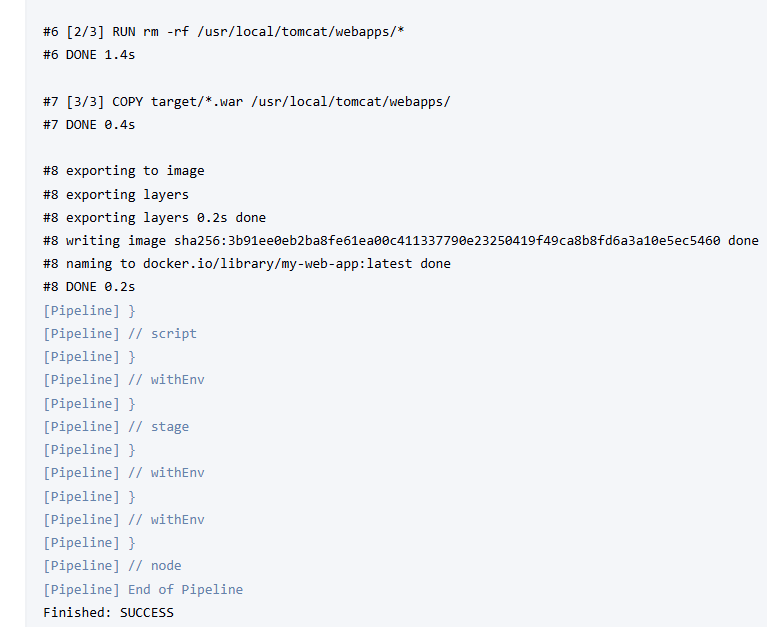
}

}

}

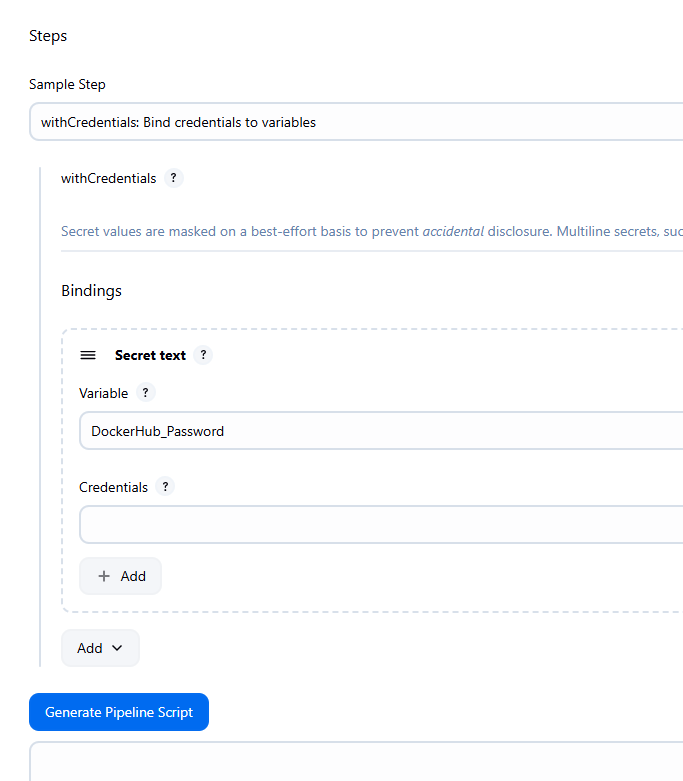


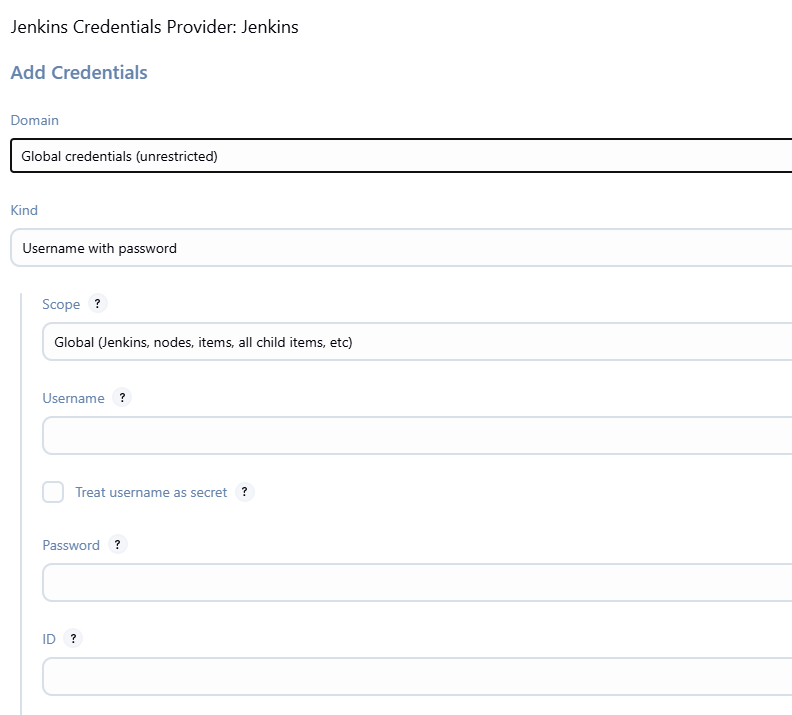




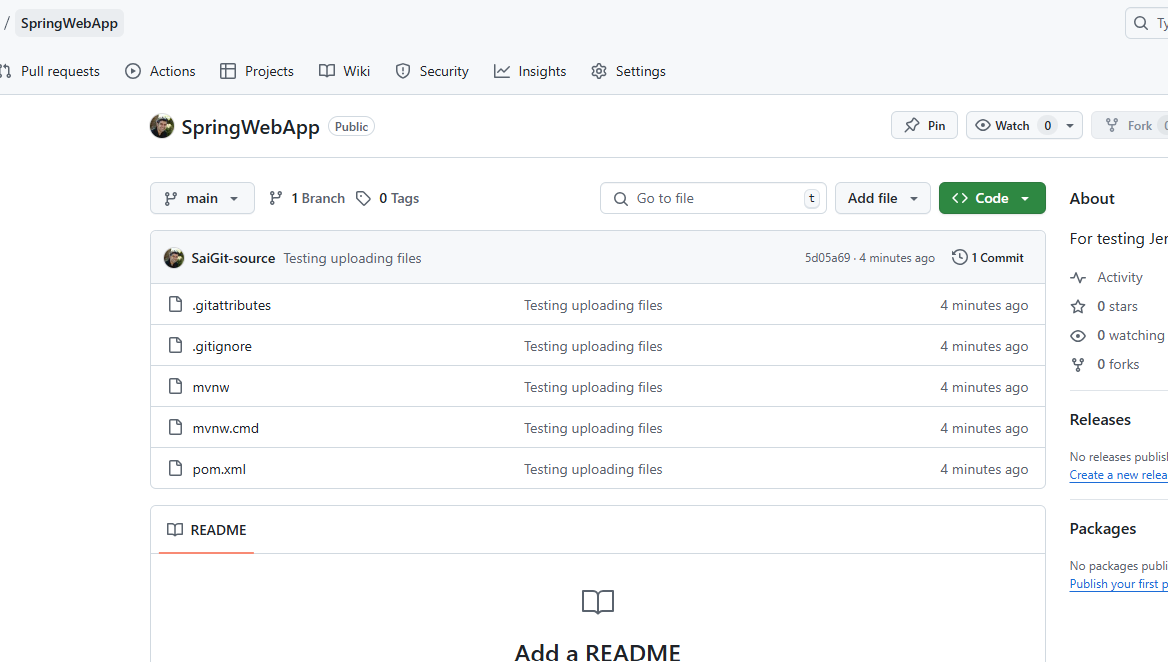
Now to push this image we got to add credentials

Generate Pipeline Script





I have pushed my project into my Repo



Generate Pipeline Script

withCredentials([string(credentialsId: 'My-Docker-Git-pwd', variable: 'Docker\_Hub\_PWD')]) {

// some block

}

pipeline {

agent any

environment {

IMAGE\_NAME = "my-web-app"

DOCKER\_TAG = "latest"

}

tools {

maven "maven-3.9.10"

}

stages {

stage('git clone') {

steps {

git branch: 'main', url: 'https://github.com/SaiGit-source/SpringWebApp.git'

}

}

stage('demo message') {

steps {

telusko()

}

}

stage('maven build') {

steps {

sh 'mvn clean compile test package'

}

}

stage('Find WAR file') {

steps {

script {

echo "Searching for WAR file..."

def warPath = sh(

script: "find target -name '\*.war' | head -n 1",

returnStdout: true

).trim()

if (warPath) {

echo "✅ WAR file found: ${warPath}"

} else {

error("❌ WAR file not found!")

}

}

}

}

stage('Build Docker Image') {

steps {

script {

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

# Use an official Tomcat base image

FROM tomcat:11-jdk21

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/

# Expose port

EXPOSE 8080

'''

echo "✅ Dockerfile generated"

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

}

}

}

stage('Docker push') {

steps {

withCredentials([string(credentialsId: 'My-Docker-Git-pwd', variable: 'Docker\_Hub\_PWD')]) {

sh 'docker login -u SaiGit-Source -p ${Docker\_Hub\_PWD}'

sh 'docker tag ${IMAGE\_NAME}:${DOCKER\_TAG} SaiGit-Source/${IMAGE\_NAME}:${DOCKER\_TAG}'

sh 'docker push SaiGit-Source/${IMAGE\_NAME}:${DOCKER\_TAG}'

}

}

}

}

}

pipeline {

agent any

environment {

IMAGE\_NAME = "my-web-app"

DOCKER\_TAG = "latest"

}

tools {

maven "maven-3.9.10"

}

stages {

stage('git clone') {

steps {

git branch: 'main', url: 'https://github.com/SaiGit-source/SpringWebApp.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}'

}

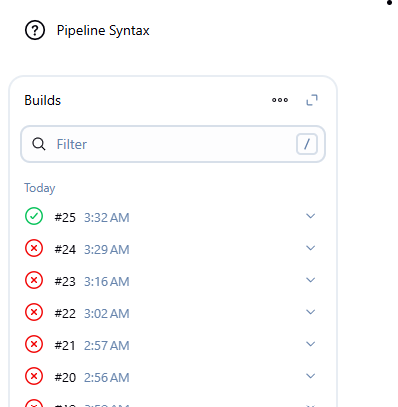
}

}

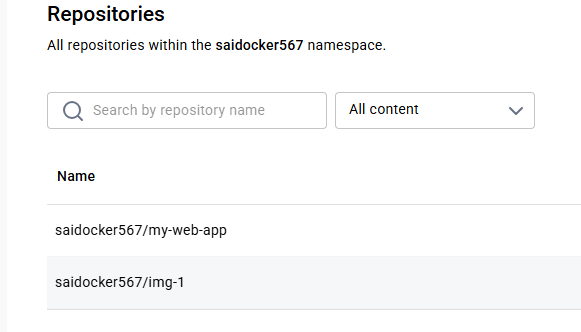
}

}

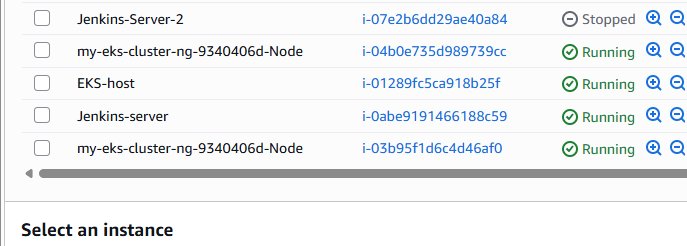
Build succeeded after numerous attempts



If we log into Docker Hub we see the image



Go to EC2 and we can see EKS clusters created



1:45

Run on EKS host

kubectl delete all --all

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

Terminated Jenkins-Server-2

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

8 ==> Update EKS Cluster Config File in Jenkins Server

Execute the below command in EKS Management Machine and Copy config file data

$ cat .kube/config

Connect to Jenkins server execute the following command to add config file into Jenkins server

$ cd /var/lib/jenkins

$ sudo mkdir .kube

$ sudo vi .kube/config

( paste config file data copied from eks host machine )

Check eks nodes

$ kubectl get nodes

$ cd ~

$ ls -la

$ sudo vi .kube/config

( paste config file data copied from eks host machine )

$ kubectl get nodes

EKS-host VM

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

apiVersion: v1

clusters:

- cluster:

certificate-authority-data: 

server: https://A4534B05E0A24CC4222E7AEBE8F2A919.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

All on Jenkins server

ubuntu@ip-172-31-11-116:~/apache-tomcat-11.0.8$ kubectl version --client

Client Version: v1.33.2

Kustomize Version: v5.6.0

ubuntu@ip-172-31-11-116:~/apache-tomcat-11.0.8$ cd /var/lib/jenkins

ubuntu@ip-172-31-11-116:/var/lib/jenkins$

All in Jenkins server

ubuntu@ip-172-31-11-116:/var/lib/jenkins$

ubuntu@ip-172-31-11-116:/var/lib/jenkins$

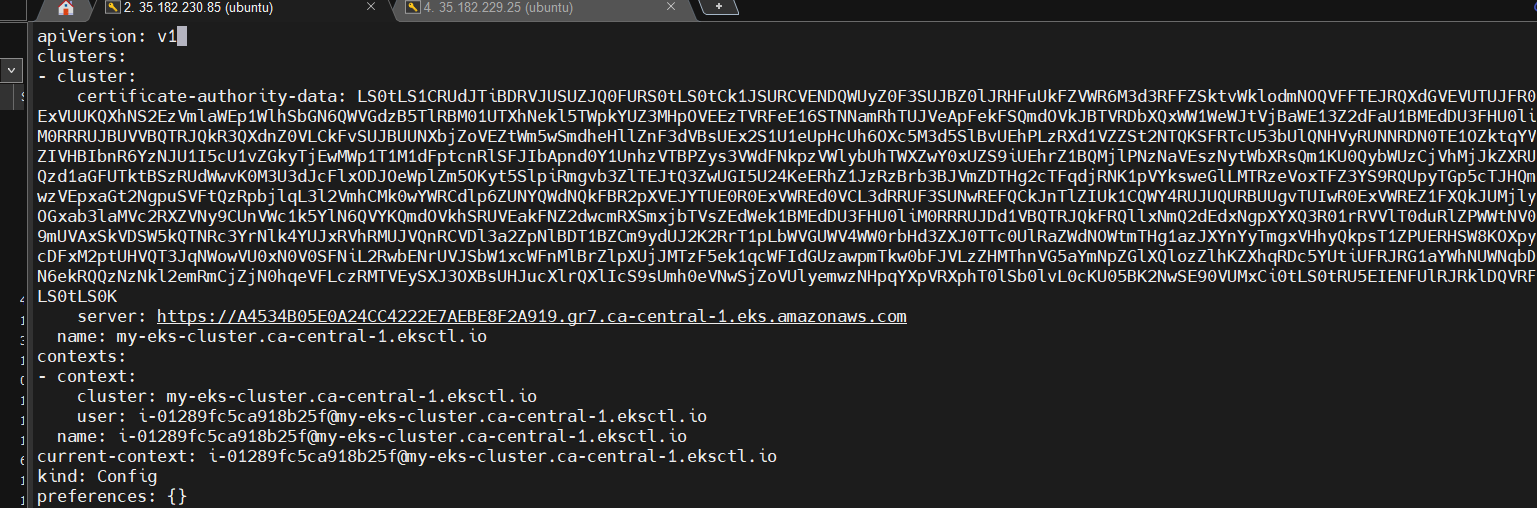
ubuntu@ip-172-31-11-116:/var/lib/jenkins$

ubuntu@ip-172-31-11-116:/var/lib/jenkins$ sudo mkdir .kube

ubuntu@ip-172-31-11-116:/var/lib/jenkins$ cd .kube/

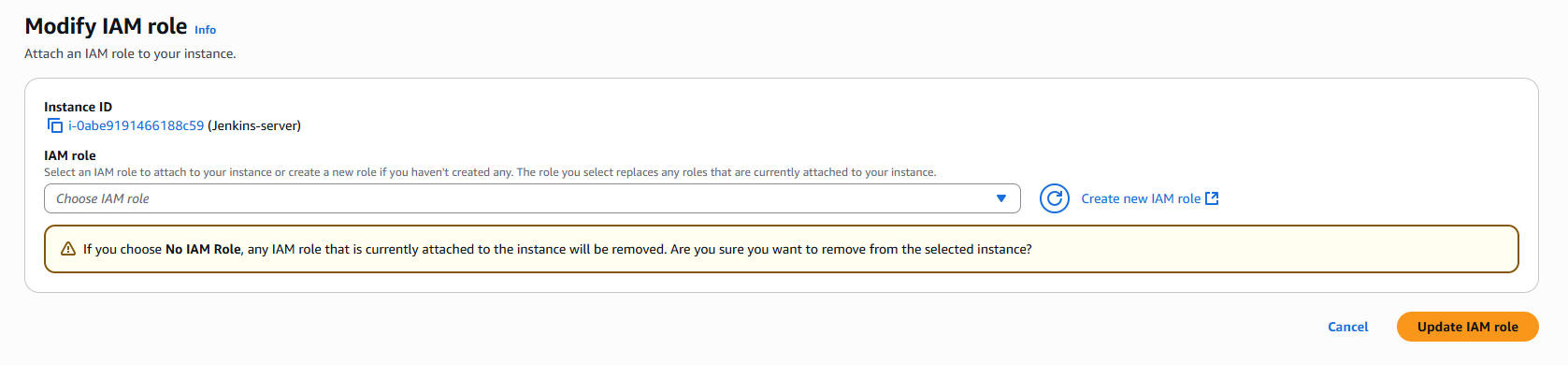
ubuntu@ip-172-31-11-116:/var/lib/jenkins/.kube$ sudo vi config

Copy paste contents from config file from EKS-Host



Esc -> :wq

===> Attach created role to Jenkins Server host vm --> Actions-> Security-> Modify IAM role --> add created iam role



eks-role is already there

Update IAM role

Copy paste the config content into this config file

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

ubuntu@ip-172-31-11-116:~/.kube$ pwd

/home/ubuntu/.kube

Path is in home

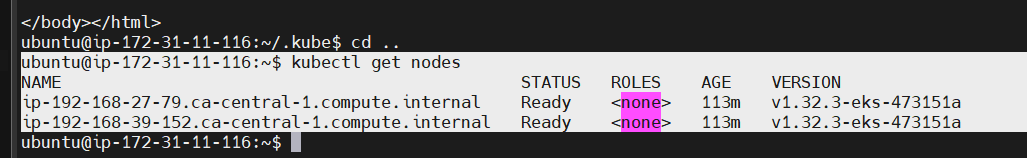
All in Jenkins Server

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

NAME STATUS ROLES AGE VERSION

ip-192-168-27-79.ca-central-1.compute.internal Ready <none> 113m v1.32.3-eks-473151a

ip-192-168-39-152.ca-central-1.compute.internal Ready <none> 113m v1.32.3-eks-473151a



1:56

For adding K8s into Pipeline script

pipeline {

agent any

environment {

IMAGE\_NAME = "my-web-app"

DOCKER\_TAG = "latest"

}

tools {

maven "maven-3.9.10"

}

stages {

stage('git clone') {

steps {

git branch: 'main', url: 'https://github.com/SaiGit-source/SpringWebApp.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('Deploy to Kubernetes') {

steps {

script {

// Write Kubernetes Deployment YAML

writeFile file: 'deployment.yaml', text: '''

apiVersion: apps/v1

kind: Deployment

metadata:

name: web-app-deployment

spec:

replicas: 2

selector:

matchLabels:

app: web-app

template:

metadata:

labels:

app: web-app

spec:

containers:

- name: web-container

image: saidocker567/my-web-app:latest

ports:

- containerPort: 8080

'''

// Write Kubernetes Service YAML

writeFile file: 'service.yaml', text: '''

apiVersion: v1

kind: Service

metadata:

name: web-app-service

spec:

type: LoadBalancer

selector:

app: web-app

ports:

- protocol: TCP

port: 80

targetPort: 8080

'''

// Apply Kubernetes manifests

sh 'kubectl apply -f deployment.yaml'

sh 'kubectl apply -f service.yaml'

}

}

}

}

We have to write a K8s manifest file in the pipeline similar to Dockerfile

<https://github.com/SaiGit-source/SpringWebApp>

This part could be avoided if we already have a K8s manifest file in the Github repo since I don’t have the manifest file I have to create one in the pipeline itself

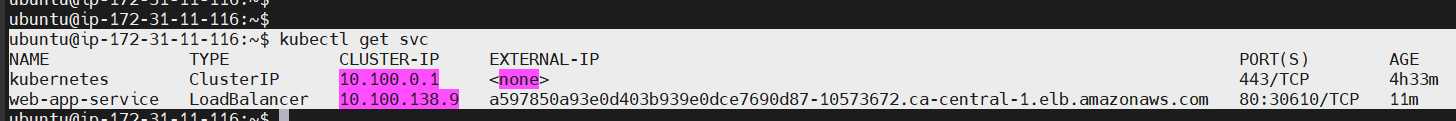


ubuntu@ip-172-31-11-116:~$ kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 4h33m

web-app-service LoadBalancer 10.100.138.9 a597850a93e0d403b939e0dce7690d87-10573672.ca-central-1.elb.amazonaws.com 80:30610/TCP 11m



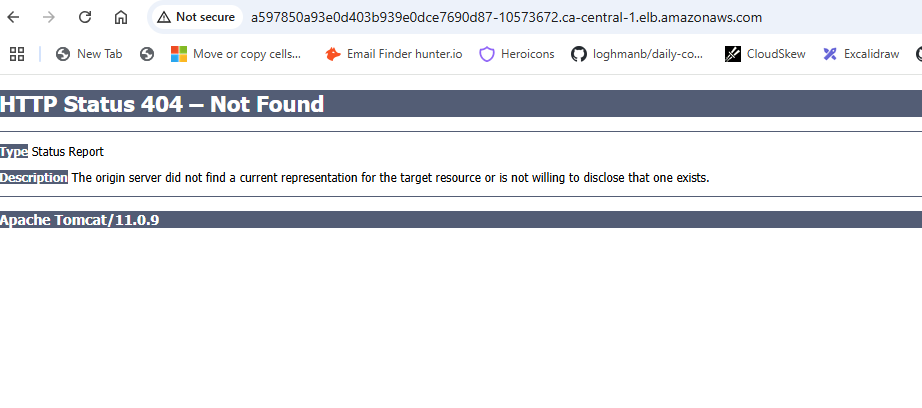
Copy this

a597850a93e0d403b939e0dce7690d87-10573672.ca-central-1.elb.amazonaws.com

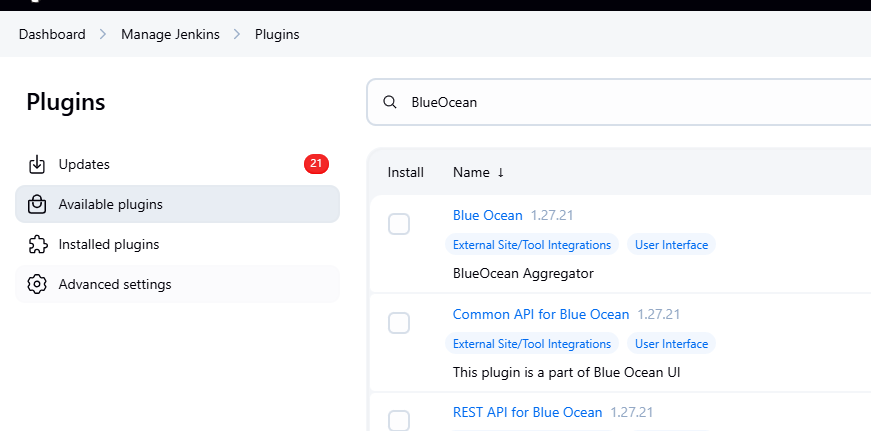
In the application controller, we have only ‘/’ as endpoint



Application is deployed but not opening



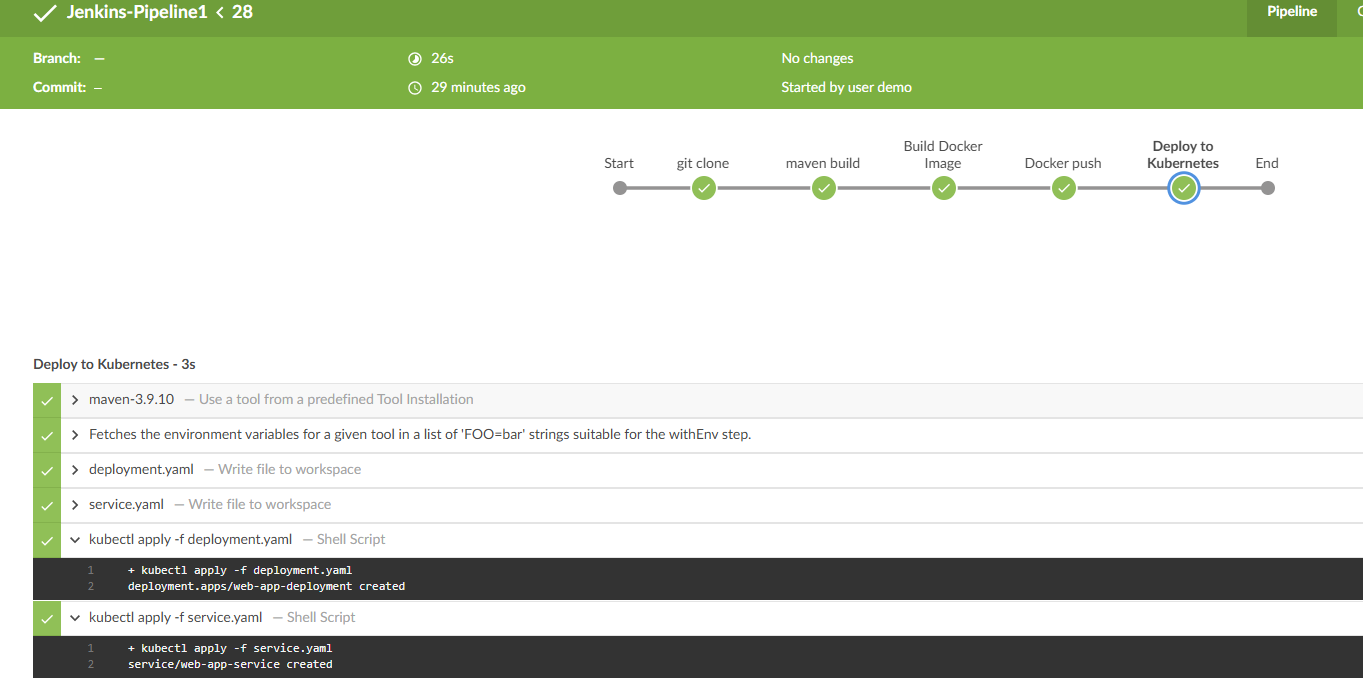
Look for BlueOcean, we can check the Logs from each stage and see what failed the App



Click Install

Click on Dashboard

Open Blue Ocean



Second way to deploy pipeline

pipeline {

agent any

environment {

IMAGE\_NAME = "my-web-app"

DOCKER\_TAG = "latest"

}

tools {

maven "maven"

}

stages {

stage('git clone') {

steps {

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

}

}

stage('maven build') {

steps {

sh 'mvn clean compile test package'

}

}

stage('Build Docker Image') {

steps {

script {

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

}

}

}

stage('Docker Push') {

steps {

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

sh "docker login -u haidertelusko -p ${Docker\_Hub\_PWD}"

sh "docker tag ${IMAGE\_NAME}:${DOCKER\_TAG} haidertelusko/${IMAGE\_NAME}:${DOCKER\_TAG}"

sh "docker push haidertelusko/${IMAGE\_NAME}:${DOCKER\_TAG}"

}

}

}

stage('k8s - deployment') {

steps {

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

}

}

}

}