Complete CI/CD Flow

Tools are used:

* Ubuntu
* Terraform
* Git
* Github
* Jenkins
* Docker
* Dockerhub
* SonarQube
* Trivy
* OWASP(NA)
* EKS
* ArgoCD(GITOPS)

Terraform to launch EC2(Jenkins) instance with pre-requisites:

Note : Install **aws cli** and **aws configure** to set **Accesskey and Secretkey** and add **Elastic IP** to the Jenkins server**(Optional but in Production/Dev is must)**

**Terraform code:**

provider.tf

========

provider "aws" {

region = "us-west-2"

}

===========

main.tf

=======

#Vpc

module "vpc" {

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

name = "jenkins\_vpc"

cidr = var.vpc\_cidr

azs = data.aws\_availability\_zones.azs.names

public\_subnets = var.public\_subnets

enable\_dns\_hostnames = true

map\_public\_ip\_on\_launch = true

tags = {

Name = "jenkins\_vpc"

Terraform = "true"

Environment = "dev"

}

public\_subnet\_tags = {

Name = "jenkins\_subnet"

}

}

#sg

module "sg" {

source = "terraform-aws-modules/security-group/aws"

name = "jenkins\_sg"

description = "Security group for jenkins server"

vpc\_id = module.vpc.vpc\_id

ingress\_with\_cidr\_blocks = [

{

from\_port = 0

to\_port = 0

protocol = "-1"

description = "HTTP"

cidr\_blocks = "0.0.0.0/0"

},

{

from\_port = 22

to\_port = 22

protocol = "tcp"

description = "SSH"

cidr\_blocks = "0.0.0.0/0"

}

]

egress\_with\_cidr\_blocks = [

{

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = "0.0.0.0/0"

}

]

tags = {

Name = "jenkins\_sg"

}

}

#ec2

module "ec2\_instance" {

source = "terraform-aws-modules/ec2-instance/aws"

name = "jenkins\_server"

instance\_type = var.instance\_type

ami = data.aws\_ami.example.id

key\_name = "ayush2"

monitoring = true

vpc\_security\_group\_ids = [module.sg.security\_group\_id]

subnet\_id = module.vpc.public\_subnets[0]

associate\_public\_ip\_address = true

availability\_zone = data.aws\_availability\_zones.azs.names[0]

user\_data = file("jenkins-install.sh")

tags = {

Name = "jankins\_server"

Terraform = "true"

Environment = "dev"

}

}

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

variable.tf

===========

variable "vpc\_cidr" {

description = "Vpc CIDR"

type = string

}

variable "public\_subnets" {

description = "public\_subnets CIDR"

type = list(string)

}

variable "instance\_type" {

description = "Instance Type"

type = string

}

=========

backend.tf

========

terraform {

backend "s3" {

bucket = "testayush"

key = "jenkins/terraform.tfstate"

region = "us-west-2"

}

}

===========

data.tf

=========

data "aws\_ami" "example" {

most\_recent = true

owners = ["amazon"]

filter {

name = "name"

values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20231207"]

}

filter {

name = "root-device-type"

values = ["ebs"]

}

filter {

name = "virtualization-type"

values = ["hvm"]

}

}

data "aws\_availability\_zones" "azs" {}

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

jenkins-install.sh

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

#!/bin/bash

# For Ubuntu 22.04

# Intsalling Java

sudo apt update -y

sudo apt install openjdk-17-jre -y

sudo apt install openjdk-17-jdk -y

java --version

# Installing Jenkins

curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \

/usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

https://pkg.jenkins.io/debian binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update -y

sudo apt-get install jenkins -y

# Installing Docker

sudo apt update -y

sudo apt install docker.io -y

sudo usermod -aG docker jenkins

sudo usermod -aG docker ubuntu

sudo systemctl restart docker

sudo chmod 777 /var/run/docker.sock

# If you don't want to install Jenkins, you can create a container of Jenkins

# docker run -d -p 8080:8080 -p 50000:50000 --name jenkins-container jenkins/jenkins:lts

# Run Docker Container of Sonarqube

#docker run -d --name sonar -p 9000:9000 sonarqube:lts-community

docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube

# Installing AWS CLI

#!/bin/bash

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

sudo apt install unzip -y

unzip awscliv2.zip

sudo ./aws/install

# Installing Kubectl

#!/bin/bash

sudo apt update

sudo apt install curl -y

sudo curl -LO "https://dl.k8s.io/release/v1.28.4/bin/linux/amd64/kubectl"

sudo chmod +x kubectl

sudo mv kubectl /usr/local/bin/

kubectl version --client

# Installing eksctl

#! /bin/bash

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

sudo mv /tmp/eksctl /usr/local/bin

eksctl version

# Installing Terraform

#!/bin/bash

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

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

sudo apt update

sudo apt install terraform -y

# Installing Trivy

#!/bin/bash

sudo apt-get install wget apt-transport-https gnupg lsb-release -y

wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -

echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb\_release -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list

sudo apt update

sudo apt install trivy -y

# Intalling Helm

#! /bin/bash

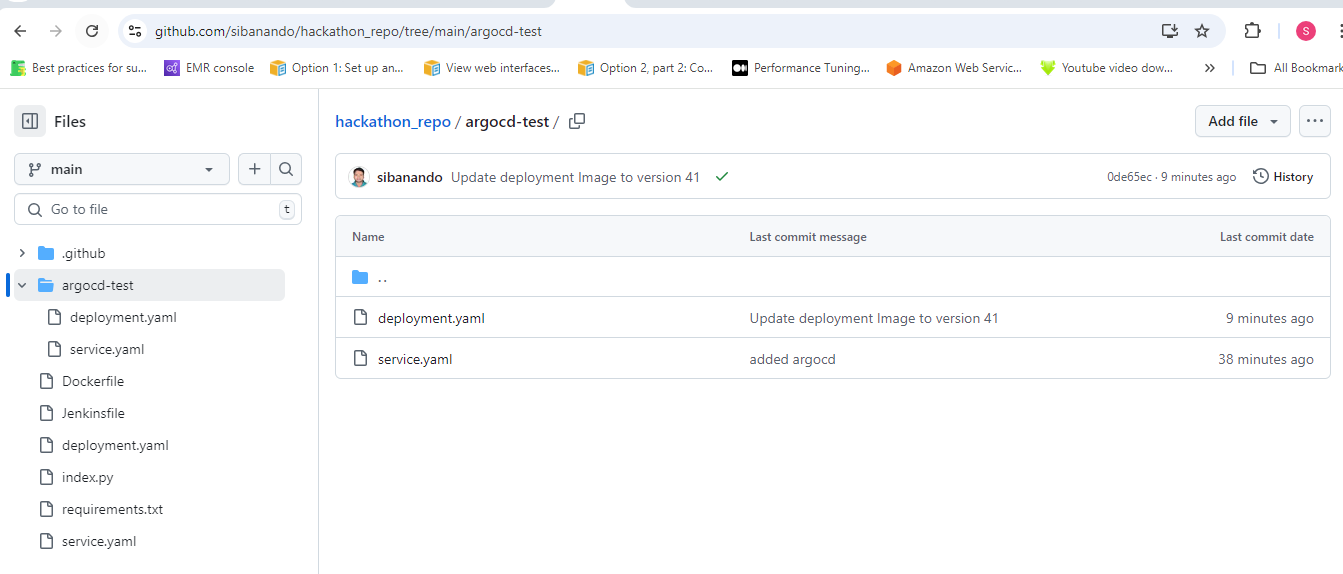
sudo snap install helm --classic

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

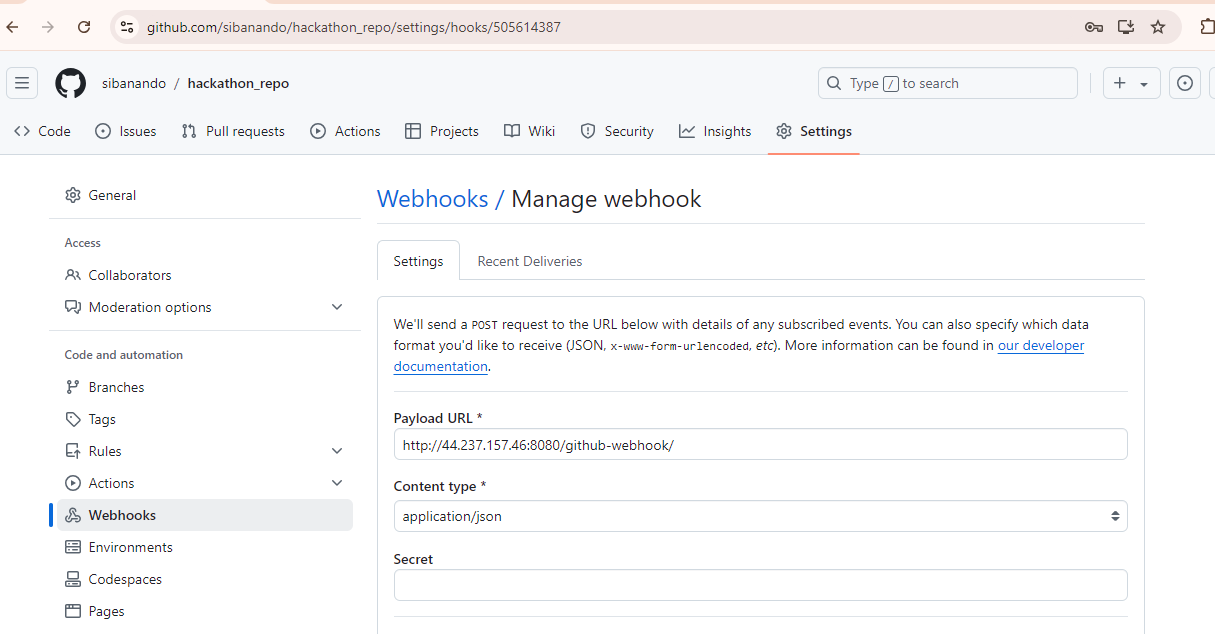
**Github**

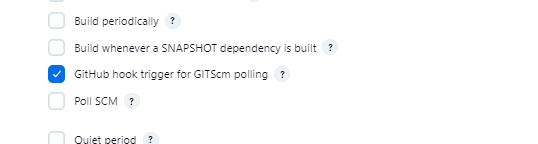
===============

Repo : <https://github.com/sibanando/hackathon_repo.git>



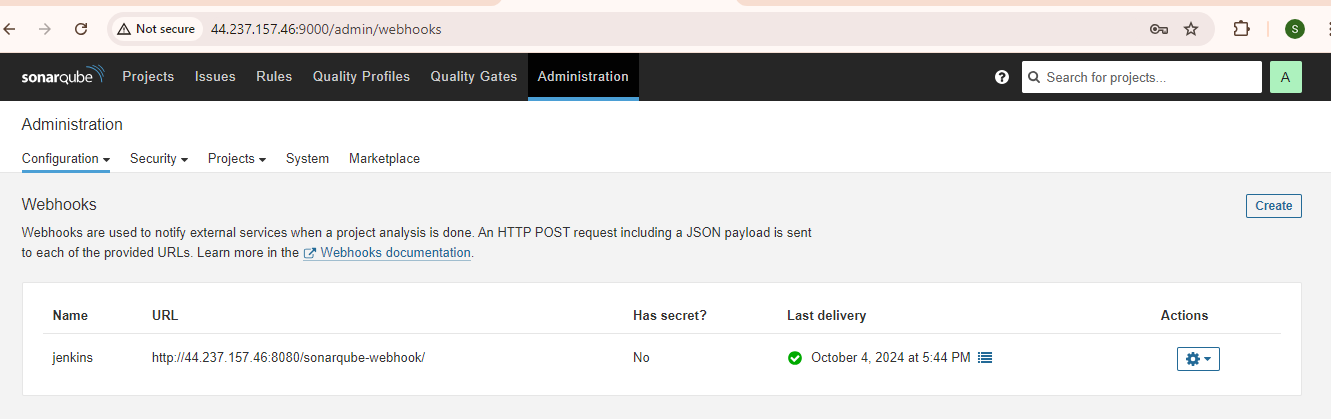
**Github webhook**

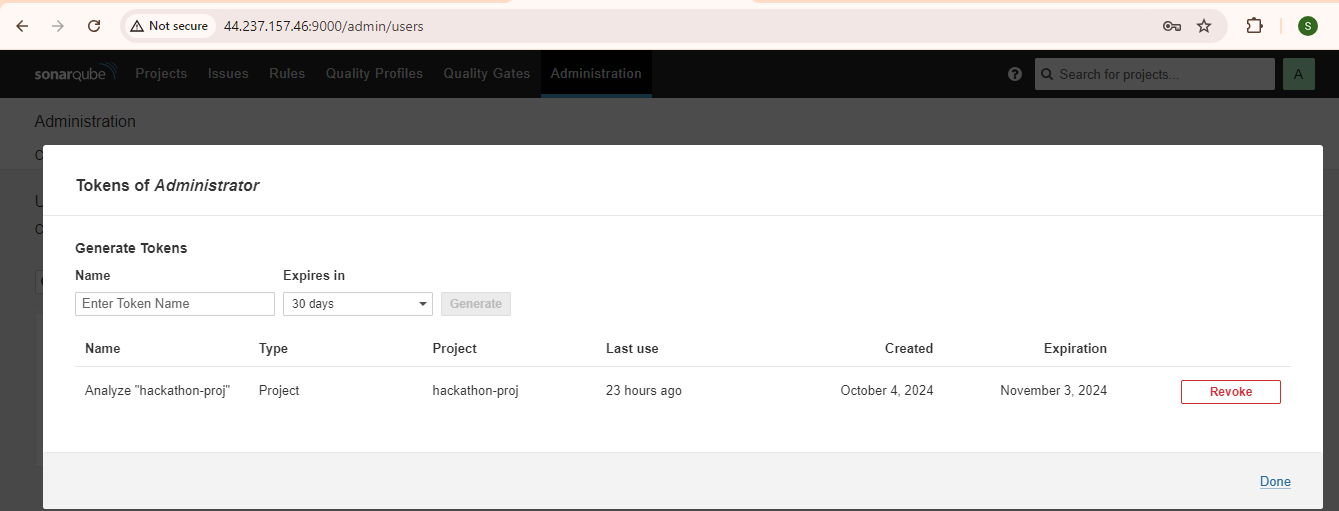


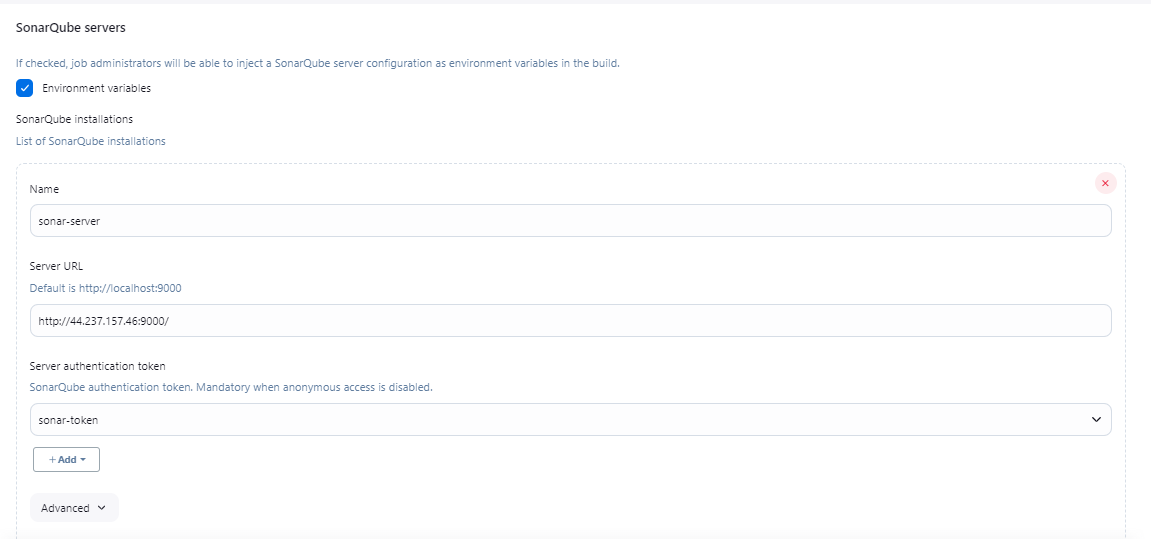


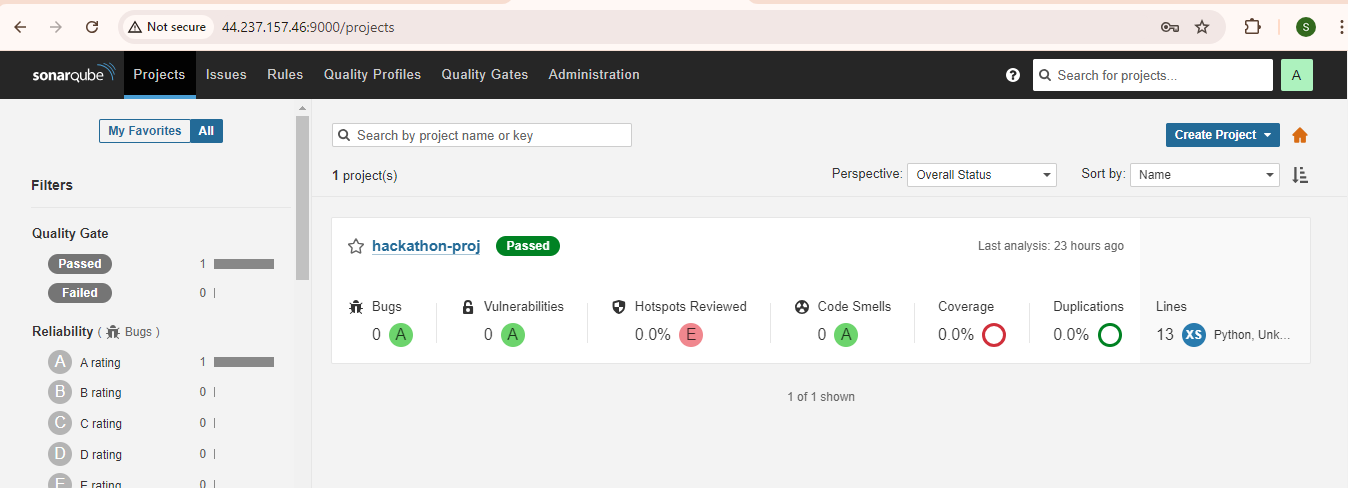
SonarQube:

docker run -d --name sonar -p 9000:9000 sonarqube:lts-community

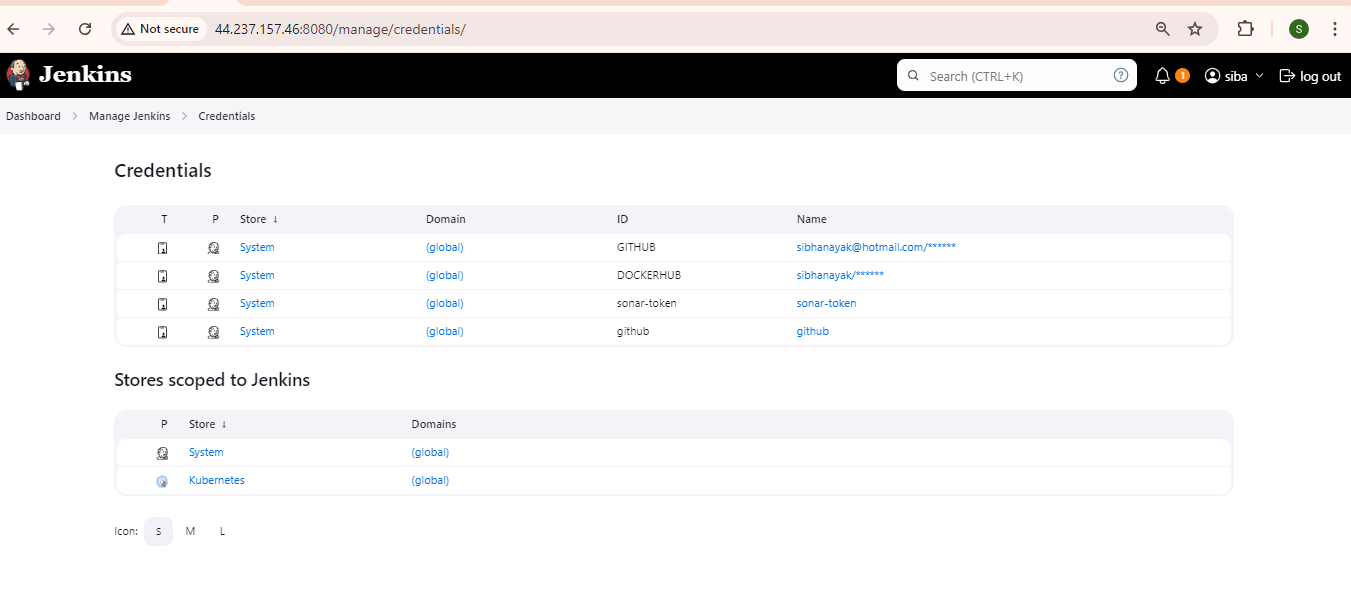








In Jenkins -> manage jenkins-> Credential



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

Jenkinsfile

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

pipeline {

agent any

environment {

DOCKERHUB\_CREDENTIALS= credentials('DOCKERHUB')

}

stages{

stage('Cleaning Workspace') {

steps {

cleanWs()

}

}

stage('Checkout from Git') {

steps {

git branch: 'main', credentialsId: 'github', url: 'https://github.com/sibanando/hackathon\_repo.git'

}

}

stage('Trivy File Scan') {

steps {

sh 'trivy fs . > trivyfs.txt'

}

}

stage('SAST - Sonar') {

environment {

scannerHome = tool 'sonar-scanner';

}

steps {

withSonarQubeEnv(credentialsId: 'sonar-token', installationName: 'sonar-server') {

sh "${scannerHome}/bin/sonar-scanner -Dsonar.projectName=hackathon-proj -Dsonar.projectKey=hackathon-proj "

}

}

}

stage('Quality Check') {

steps {

script {

waitForQualityGate abortPipeline: false, credentialsId: 'sonar-token'

}

}

}

// stage('OWASP Dependency-Check Scan') {

// steps {

//

// dependencyCheck additionalArguments: '--scan ./ --disableYarnAudit --disableNodeAudit', odcInstallation: 'DP-Check'

// dependencyCheckPublisher pattern: '\*\*/dependency-check-report.xml'

//

// }

// }

// Building Docker images

// Uploading Docker images into Docker Hub

stage('Building image') {

steps{

script {

sh 'docker system prune -f'

sh 'docker container prune -f'

sh 'docker build -t sibhanayak/pythonapp:$BUILD\_NUMBER .'

}

}

}

stage('Login to Docker Hub') {

steps{

sh 'echo $DOCKERHUB\_CREDENTIALS\_PSW | docker login -u $DOCKERHUB\_CREDENTIALS\_USR --password-stdin'

echo 'Login Completed'

}

}

// Running Docker container, make sure port 8096 is opened in

stage('Docker Push') {

steps{

script {

sh 'docker push sibhanayak/pythonapp:$BUILD\_NUMBER'

}

}

}

stage("TRIVY Image Scan") {

steps {

sh 'trivy image sibhanayak/pythonapp:$BUILD\_NUMBER > trivyimage.txt'

}

}

stage('Checkout2 from Git') {

steps {

git branch: 'main', credentialsId: 'github', url: 'https://github.com/sibanando/hackathon\_repo.git'

}

}

stage('Update Deployment file') {

environment {

GIT\_REPO\_NAME = "hackathon\_repo"

GIT\_USER\_NAME = "sibanando"

}

steps {

withCredentials([string(credentialsId: 'github', variable: 'GITHUB\_TOKEN')]) {

sh '''

git config user.email "sibhanayak@hotmail.com"

git config user.name "sibanando"

BUILD\_NUMBER=${BUILD\_NUMBER}

echo $BUILD\_NUMBER

sed -i "s/replaceImageTag/${BUILD\_NUMBER}/g" argocd-test/deployment.yaml

git add argocd-test/deployment.yaml

git commit -m "Update deployment Image to version \${BUILD\_NUMBER}"

git push https://${GITHUB\_TOKEN}@github.com/${GIT\_USER\_NAME}/${GIT\_REPO\_NAME} HEAD:main

'''

}

}

}

}

post{

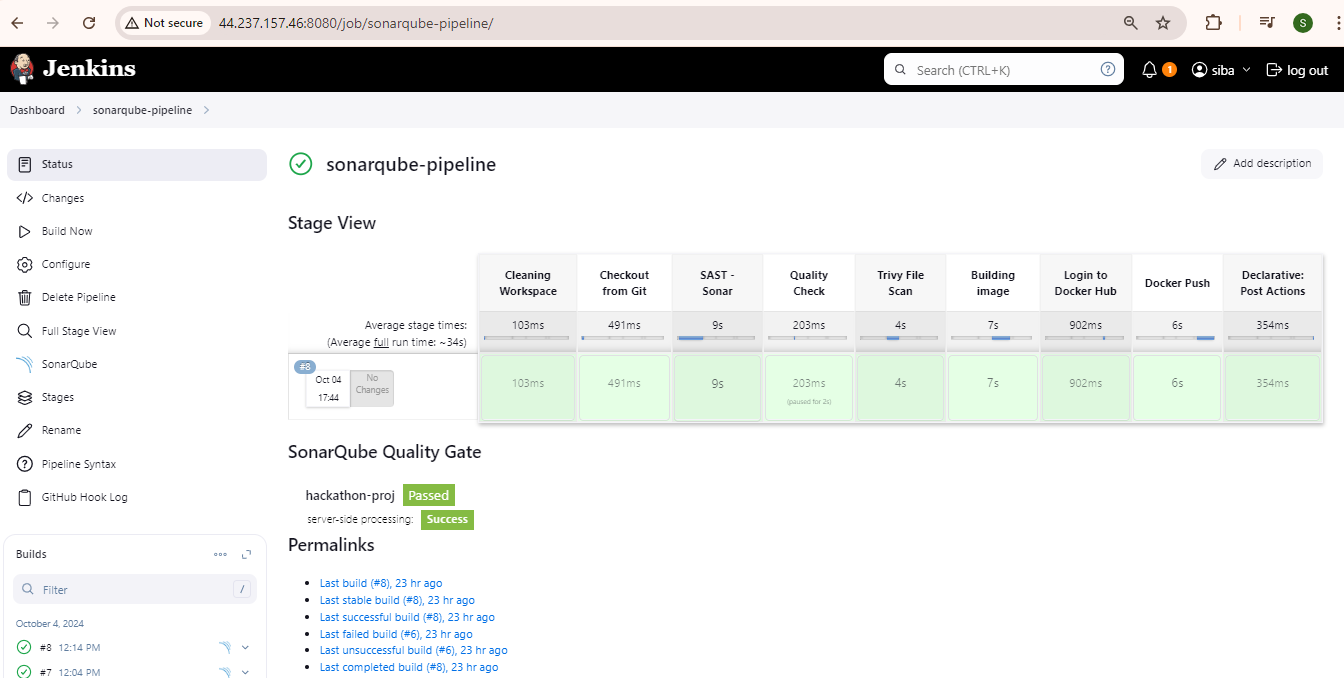
always {

sh 'docker logout'

}

}

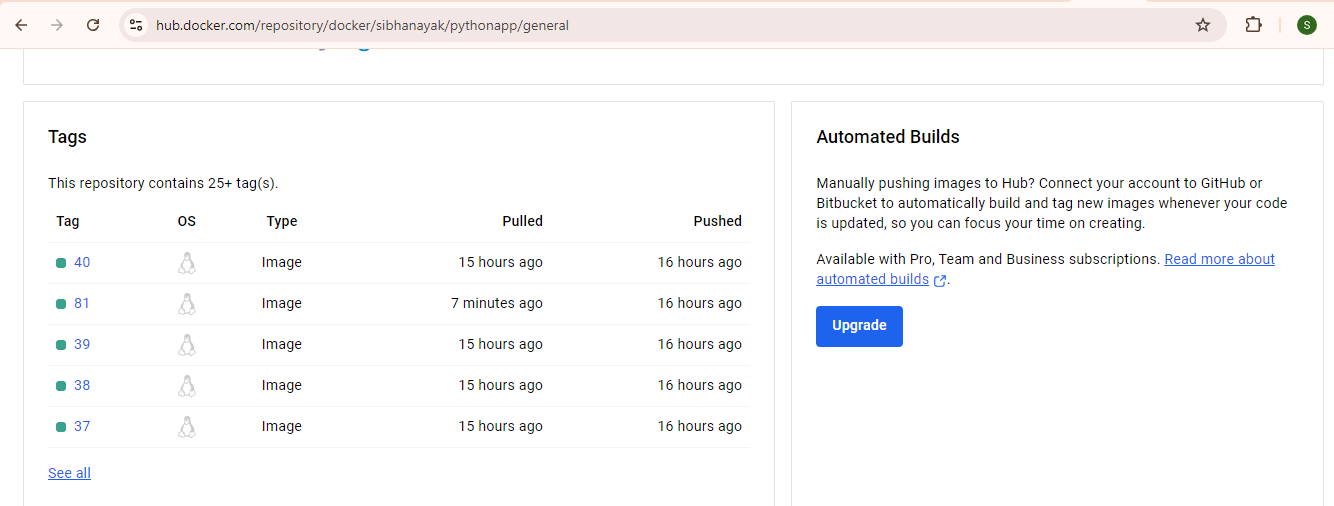
}



===========

Dockerhub

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



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

EKS cluster

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

**Creating eks:**

eksctl create cluster --name hackathon-k8s --region us-west-2 --node-type t2.medium --zones us-west-2a,us-west-2b

**Update-kubeconfig to access Kubernetes in kubectl :**

aws eks update-kubeconfig --region us-west-2 --name hackathon-k8s

**Delete Kubernetes cluster**

eksctl delete cluster --name hackathon-k8s --region us-west-2

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

ARGOCD

===========

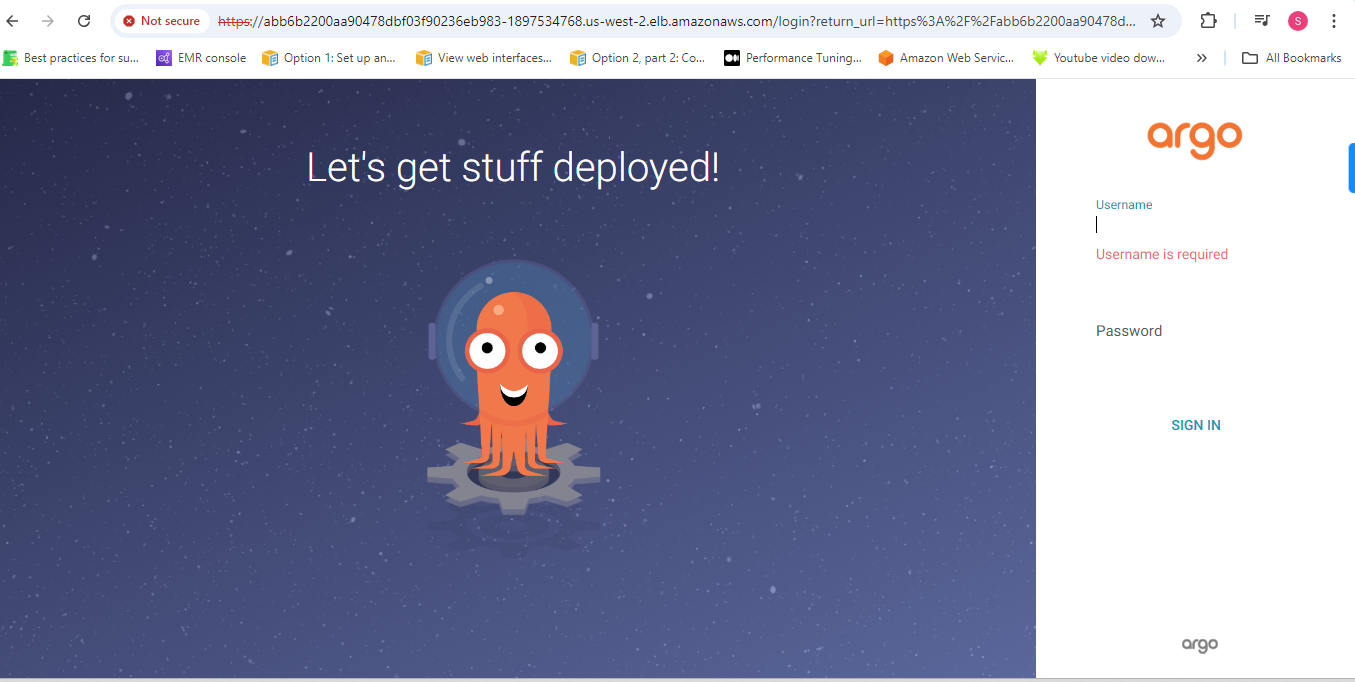
kubectl create namespace argocd

kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

kubectl get pods -n argocd -w

kubectl edit svc argocd-server -n argocd

kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'



ubuntu@ip-10-0-1-36:~$ kubectl get secret -n argocd

NAME TYPE DATA AGE

argocd-initial-admin-secret Opaque 1 4h

argocd-notifications-secret Opaque 0 4h

argocd-redis Opaque 1 4h

argocd-secret Opaque 5 4h

kubectl edit secret argocd-initial-admin-secret -n argocd

echo WVlZd24tWUVZbFdCdTBCUw== |base64 –decode

