**SESSION - 84**

--> vpc

--> install the pluging everything

--> install the shared libraries

--> let us create the all the pipelines

--> catalogue,user.cart,shipping,payment everything will create through shared library.

--> cd terraform-aws-eks

--> cd 30-ecr

--> terraform plan

--> cd jenkins-shared-library

--> push the code

--> Manage jenkins

--> Name: jenkins library

--> Default version : main

--> Source code management: git

--> Project repository : <https://github.com/daws-84s/jenkins-shared-library.git>

--> Welcome to jenkins

--> New item : ROBOSHOP --> Folder --> inside this : new item: catalogue-ci

--> multi branch pipeline

--> catalogue-ci

--> console output

--> rebuild the code

--> catalogue-ci success

--> next user

--> create user-ci in jenkins

--> then cart

--> cart also same like catalogue and user

--> git checkout -b feature

**vars/javaEKSPipeline.groovy**

def call(Map configMap){

pipeline {

agent {

label 'AGENT-1'

}

environment {

appVersion = ''

REGION = "us-east-1"

ACC\_ID = "160885265516"

PROJECT = configMap.get('project')

COMPONENT = configMap.get('component')

}

options {

timeout(time: 30, unit: 'MINUTES')

disableConcurrentBuilds()

}

parameters {

booleanParam(name: 'deploy', defaultValue: false, description: 'Toggle this value')

}

// Build

stages {

stage('Read pom.xml') {

steps {

script {

appVersion = readMavenPom().getVersion()

echo "app version: ${appVersion}"

}

}

}

stage('Install Dependencies') {

steps {

script {

sh """

mvn clean package

"""

}

}

}

stage('Unit Testing') {

steps {

script {

sh """

echo "unit tests"

"""

}

}

}

/\* stage('Sonar Scan') {

environment {

scannerHome = tool 'sonar-7.2'

}

steps {

script {

// Sonar Server envrionment

withSonarQubeEnv(installationName: 'sonar-7.2') {

sh "${scannerHome}/bin/sonar-scanner"

}

}

}

} \*/

// Enable webhook in sonarqube server and wait for results

/\* stage("Quality Gate") {

steps {

timeout(time: 1, unit: 'HOURS') {

waitForQualityGate abortPipeline: true }

}

}

stage('Check Dependabot Alerts') {

environment {

GITHUB\_TOKEN = credentials('github-token')

}

steps {

script {

// Fetch alerts from GitHub

def response = sh(

script: """

curl -s -H "Accept: application/vnd.github+json" \

-H "Authorization: token ${GITHUB\_TOKEN}" \

https://api.github.com/repos/daws-84s/catalogue/dependabot/alerts

""",

returnStdout: true

).trim()

// Parse JSON

def json = readJSON text: response

// Filter alerts by severity

def criticalOrHigh = json.findAll { alert ->

def severity = alert?.security\_advisory?.severity?.toLowerCase()

def state = alert?.state?.toLowerCase()

return (state == "open" && (severity == "critical" || severity == "high"))

}

if (criticalOrHigh.size() > 0) {

error "❌ Found ${criticalOrHigh.size()} HIGH/CRITICAL Dependabot alerts. Failing pipeline!"

} else {

echo "✅ No HIGH/CRITICAL Dependabot alerts found."

}

}

}

}\*/

stage('Docker Build') {

steps {

script {

withAWS(credentials: 'aws-creds', region: 'us-east-1') {

sh """

aws ecr get-login-password --region ${REGION} | docker login --username AWS --password-stdin ${ACC\_ID}.dkr.ecr.us-east-1.amazonaws.com

docker build -t ${ACC\_ID}.dkr.ecr.us-east-1.amazonaws.com/${PROJECT}/${COMPONENT}:${appVersion} .

docker push ${ACC\_ID}.dkr.ecr.us-east-1.amazonaws.com/${PROJECT}/${COMPONENT}:${appVersion}

#aws ecr wait image-scan-complete --repository-name ${PROJECT}/${COMPONENT} --image-id imageTag=${appVersion} --region ${REGION}

"""

}

}

}

}

/\* stage('Check Scan Results') {

steps {

script {

withAWS(credentials: 'aws-creds', region: 'us-east-1') {

// Fetch scan findings

def findings = sh(

script: """

aws ecr describe-image-scan-findings \

--repository-name ${PROJECT}/${COMPONENT} \

--image-id imageTag=${appVersion} \

--region ${REGION} \

--output json

""",

returnStdout: true

).trim()

// Parse JSON

def json = readJSON text: findings

def highCritical = json.imageScanFindings.findings.findAll {

it.severity == "HIGH" || it.severity == "CRITICAL"

}

if (highCritical.size() > 0) {

echo "❌ Found ${highCritical.size()} HIGH/CRITICAL vulnerabilities!"

currentBuild.result = 'FAILURE'

error("Build failed due to vulnerabilities")

} else {

echo "✅ No HIGH/CRITICAL vulnerabilities found."

}

}

}

}

} \*/

stage('Trigger Deploy') {

when{

expression { params.deploy }

}

steps {

script {

//build job: 'catalogue-cd',

build job: "../${COMPONENT}-cd",

parameters: [

string(name: 'appVersion', value: "${appVersion}"),

string(name: 'deploy\_to', value: 'dev')

],

propagate: false, // even SG fails VPC will not be effected

wait: false // VPC will not wait for SG pipeline completion

}

}

}

}

post {

always {

echo 'I will always say Hello again!'

deleteDir()

}

success {

echo 'Hello Success'

}

failure {

echo 'Hello Failure'

}

}

}

}

**Shipping/Jenkinsfile**

@Library('jenkins-shared-library') \_

def configMap = [

project : "roboshop",

component: "shipping"

]

if( ! env.BRANCH\_NAME.equalsIgnoreCase('main') ){ // if not equals to main

javaEKSPipeline(configMap) // by default it will call, call function inside this pipeline

}

else{

echo "Please proceed with PROD process"

}

--> push the code

**Jenkins-shared-library/vars/pythonEKSPipeline.groovy**

def call(Map configMap){

pipeline {

agent {

label 'AGENT-1'

}

environment {

appVersion = ''

REGION = "us-east-1"

ACC\_ID = "160885265516"

PROJECT = configMap.get('project')

COMPONENT = configMap.get('component')

}

options {

timeout(time: 30, unit: 'MINUTES')

disableConcurrentBuilds()

}

parameters {

booleanParam(name: 'deploy', defaultValue: false, description: 'Toggle this value')

}

// Build

stages {

stage('Read version') {

steps {

script {

appVersion = readFile('version').trim()

echo "app version: ${appVersion}"

}

}

}

stage('Install Dependencies') {

steps {

script {

sh """

pip3 install -r requirements.txt

"""

}

}

}

stage('Unit Testing') {

steps {

script {

sh """

echo "unit tests"

"""

}

}

}

/\* stage('Sonar Scan') {

environment {

scannerHome = tool 'sonar-7.2'

}

steps {

script {

// Sonar Server envrionment

withSonarQubeEnv(installationName: 'sonar-7.2') {

sh "${scannerHome}/bin/sonar-scanner"

}

}

}

} \*/

// Enable webhook in sonarqube server and wait for results

/\* stage("Quality Gate") {

steps {

timeout(time: 1, unit: 'HOURS') {

waitForQualityGate abortPipeline: true }

}

}

stage('Check Dependabot Alerts') {

environment {

GITHUB\_TOKEN = credentials('github-token')

}

steps {

script {

// Fetch alerts from GitHub

def response = sh(

script: """

curl -s -H "Accept: application/vnd.github+json" \

-H "Authorization: token ${GITHUB\_TOKEN}" \

https://api.github.com/repos/daws-84s/catalogue/dependabot/alerts

""",

returnStdout: true

).trim()

// Parse JSON

def json = readJSON text: response

// Filter alerts by severity

def criticalOrHigh = json.findAll { alert ->

def severity = alert?.security\_advisory?.severity?.toLowerCase()

def state = alert?.state?.toLowerCase()

return (state == "open" && (severity == "critical" || severity == "high"))

}

if (criticalOrHigh.size() > 0) {

error "❌ Found ${criticalOrHigh.size()} HIGH/CRITICAL Dependabot alerts. Failing pipeline!"

} else {

echo "✅ No HIGH/CRITICAL Dependabot alerts found."

}

}

}

}\*/

stage('Docker Build') {

steps {

script {

withAWS(credentials: 'aws-creds', region: 'us-east-1') {

sh """

aws ecr get-login-password --region ${REGION} | docker login --username AWS --password-stdin ${ACC\_ID}.dkr.ecr.us-east-1.amazonaws.com

docker build -t ${ACC\_ID}.dkr.ecr.us-east-1.amazonaws.com/${PROJECT}/${COMPONENT}:${appVersion} .

docker push ${ACC\_ID}.dkr.ecr.us-east-1.amazonaws.com/${PROJECT}/${COMPONENT}:${appVersion}

#aws ecr wait image-scan-complete --repository-name ${PROJECT}/${COMPONENT} --image-id imageTag=${appVersion} --region ${REGION}

"""

}

}

}

}

/\* stage('Check Scan Results') {

steps {

script {

withAWS(credentials: 'aws-creds', region: 'us-east-1') {

// Fetch scan findings

def findings = sh(

script: """

aws ecr describe-image-scan-findings \

--repository-name ${PROJECT}/${COMPONENT} \

--image-id imageTag=${appVersion} \

--region ${REGION} \

--output json

""",

returnStdout: true

).trim()

// Parse JSON

def json = readJSON text: findings

def highCritical = json.imageScanFindings.findings.findAll {

it.severity == "HIGH" || it.severity == "CRITICAL"

}

if (highCritical.size() > 0) {

echo "❌ Found ${highCritical.size()} HIGH/CRITICAL vulnerabilities!"

currentBuild.result = 'FAILURE'

error("Build failed due to vulnerabilities")

} else {

echo "✅ No HIGH/CRITICAL vulnerabilities found."

}

}

}

}

} \*/

stage('Trigger Deploy') {

when{

expression { params.deploy }

}

steps {

script {

//build job: 'catalogue-cd',

build job: "../${COMPONENT}-cd",

parameters: [

string(name: 'appVersion', value: "${appVersion}"),

string(name: 'deploy\_to', value: 'dev')

],

propagate: false, // even SG fails VPC will not be effected

wait: false // VPC will not wait for SG pipeline completion

}

}

}

}

post {

always {

echo 'I will always say Hello again!'

deleteDir()

}

success {

echo 'Hello Success'

}

failure {

echo 'Hello Failure'

}

}

}

}

--> cd payment

**Payment/Jenkinsfile**

@Library('jenkins-shared-library') \_

def configMap = [

project : "roboshop",

component: "payment"

]

if( ! env.BRANCH\_NAME.equalsIgnoreCase('main') ){ // if not equals to main

pythonEKSPipeline(configMap) // by default it will call, call function inside this pipeline

}

else{

echo "Please proceed with PROD process"

}

**Payment/version**

1.0.0

--> New item

--> payment-ci

--> add payment details

--> kubectl create namespace argocd

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

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

curl -sSL -o argocd-linux-amd64 https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64

sudo install -m 555 argocd-linux-amd64 /usr/local/bin/argocd

rm argocd-linux-amd64

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

argocd admin initial-password -n argocd

--> kubectl get svc -n argocd

--> git clone argocd

--> kubectl apply -f app-of-apps.yaml

--> cd k8-roboshop/mongodb

--> kubectl apply -f mainfest.yaml

--> cd redis

--> kubectl apply -f mainfest.yaml  
--> cd mysql

--> kubectl apply -f mainfest.yaml

--> rabbitmq

--> kubectl apply -f mainfest.yaml

--> kubectl get pods -n roboshop

--> kubectl logs mysq-log

--> kubectl get pods -n roboshop

--> install k9s

--> let us deploy our catalogue

--> make sure eks allows jenkins-agent

--> configure cluster

--> create namespace

--> install DB applications

--> frontend-ci

**frontend/Jenkinsfile**

@Library('jenkins-shared-library') \_

def configMap = [

project : "roboshop",

component: "frontend"

]

if( ! env.BRANCH\_NAME.equalsIgnoreCase('main') ){ // if not equals to main

nodejsEKSPipeline(configMap) // by default it will call, call function inside this pipeline

}

else{

echo "Please proceed with PROD process"

}

**eks-argocd/roboshop/user.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: user

namespace: roboshop

labels:

component: user

project: roboshop

tier: app

data:

MONGO\_URL: "mongodb://mongodb:27017/users"

REDIS\_URL: "redis://redis:6379"

MONGO: "true"

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: user

namespace: roboshop

# deployment labels

labels:

component: user

project: roboshop

tier: app

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: user

project: roboshop

tier: app

# This is pod definition

template:

metadata:

labels:

component: user

project: roboshop

tier: app

spec:

containers:

- name: user

image: 160885265516.dkr.ecr.us-east-1.amazonaws.com/roboshop/user:1.0.0

envFrom:

- configMapRef:

name: user

---

apiVersion: v1

kind: Service

metadata:

name: user

namespace: roboshop

labels:

component: user

project: roboshop

tier: app

spec:

selector:

component: user

project: roboshop

tier: app

ports:

- protocol: TCP

port: 8080 # service port

targetPort: 8080 # container port

**roboshop/cart.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: cart

namespace: roboshop

labels:

component: cart

project: roboshop

tier: app

data:

REDIS\_HOST: "redis"

CATALOGUE\_HOST: "catalogue"

CATALOGUE\_PORT: "8080"

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: cart

namespace: roboshop

# deployment labels

labels:

component: cart

project: roboshop

tier: app

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: cart

project: roboshop

tier: app

# This is pod definition

template:

metadata:

labels:

component: cart

project: roboshop

tier: app

spec:

containers:

- name: cart

image: 160885265516.dkr.ecr.us-east-1.amazonaws.com/roboshop/cart:1.0.0

envFrom:

- configMapRef:

name: cart

---

apiVersion: v1

kind: Service

metadata:

name: cart

namespace: roboshop

labels:

component: cart

project: roboshop

tier: app

spec:

selector:

component: cart

project: roboshop

tier: app

ports:

- protocol: TCP

port: 8080 # service port

targetPort: 8080 # container port

**roboshop/frontend.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: nginx-conf

namespace: roboshop

labels:

component: frontend

project: roboshop

tier: web

data:

nginx.conf: |

user nginx;

worker\_processes auto;

error\_log /var/log/nginx/error.log notice;

pid /run/nginx.pid;

include /usr/share/nginx/modules/\*.conf;

events {

worker\_connections 1024;

}

http {

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

access\_log /var/log/nginx/access.log main;

sendfile on;

tcp\_nopush on;

keepalive\_timeout 65;

types\_hash\_max\_size 4096;

include /etc/nginx/mime.types;

default\_type application/octet-stream;

include /etc/nginx/conf.d/\*.conf;

server {

listen 8080;

listen [::]:8080;

server\_name \_;

root /usr/share/nginx/html;

include /etc/nginx/default.d/\*.conf;

error\_page 404 /404.html;

location = /404.html {

}

error\_page 500 502 503 504 /50x.html;

location = /50x.html {

}

location /images/ {

expires 5s;

root /usr/share/nginx/html;

try\_files $uri /images/placeholder.jpg;

}

location /api/catalogue/ { proxy\_pass http://catalogue:8080/; }

location /api/user/ { proxy\_pass http://user:8080/; }

location /api/cart/ { proxy\_pass http://cart:8080/; }

location /api/shipping/ { proxy\_pass http://shipping:8080/; }

location /api/payment/ { proxy\_pass http://payment:8080/; }

location /health {

stub\_status on;

access\_log off;

}

}

}

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: frontend

namespace: roboshop

# deployment labels

labels:

component: frontend

project: roboshop

tier: web

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: frontend

project: roboshop

tier: web

# This is pod definition

template:

metadata:

labels:

component: frontend

project: roboshop

tier: web

spec:

containers:

- name: frontend

image: joindevops/frontend:v1

volumeMounts:

- name: nginx-conf

mountPath: /etc/nginx/nginx.conf

subPath: nginx.conf

readOnly: true

volumes:

- name: nginx-conf

configMap:

name: nginx-conf

items:

- key: nginx.conf

path: nginx.conf

---

apiVersion: v1

kind: Service

metadata:

name: frontend

namespace: roboshop

labels:

component: frontend

project: roboshop

tier: web

spec:

type: LoadBalancer

selector:

component: frontend

project: roboshop

tier: web

ports:

- protocol: TCP

port: 80 # service port

targetPort: 8080 # container port

**roboshop/payment.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: payment

namespace: roboshop

labels:

component: payment

project: roboshop

tier: app

data:

CART\_HOST: "cart"

CART\_PORT: "8080"

USER\_HOST: "user"

USER\_PORT: "8080"

AMQP\_HOST: "rabbitmq"

---

apiVersion: v1

kind: Secret

metadata:

name: payment

namespace: roboshop

labels:

component: payment

project: roboshop

tier: app

type: Opaque

data:

AMQP\_USER: "cm9ib3Nob3A="

AMQP\_PASS: "cm9ib3Nob3AxMjM="

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: payment

namespace: roboshop

# deployment labels

labels:

component: payment

project: roboshop

tier: app

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: payment

project: roboshop

tier: app

# This is pod definition

template:

metadata:

labels:

component: payment

project: roboshop

tier: app

spec:

containers:

- name: payment

image: 160885265516.dkr.ecr.us-east-1.amazonaws.com/roboshop/payment:1.0.0

envFrom:

- configMapRef:

name: payment

- secretRef:

name: payment

---

apiVersion: v1

kind: Service

metadata:

name: payment

namespace: roboshop

labels:

component: payment

project: roboshop

tier: app

spec:

selector:

component: payment

project: roboshop

tier: app

ports:

- protocol: TCP

port: 8080 # service port

targetPort: 8080 # container port

**roboshop/shipping.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: shipping

namespace: roboshop

labels:

component: shipping

project: roboshop

tier: app

data:

CART\_ENDPOINT: "cart:8080"

DB\_HOST: "mysql"

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: shipping

namespace: roboshop

# deployment labels

labels:

component: shipping

project: roboshop

tier: app

spec:

replicas: 1

# These are the labels replica set use to create pod replicas, this should match pod labels

selector:

matchLabels:

component: shipping

project: roboshop

tier: app

# This is pod definition

template:

metadata:

labels:

component: shipping

project: roboshop

tier: app

spec:

containers:

- name: shipping

image: 160885265516.dkr.ecr.us-east-1.amazonaws.com/roboshop/shipping:1.0

envFrom:

- configMapRef:

name: shipping

---

apiVersion: v1

kind: Service

metadata:

name: shipping

namespace: roboshop

labels:

component: shipping

project: roboshop

tier: app

spec:

selector:

component: shipping

project: roboshop

tier: app

ports:

- protocol: TCP

port: 8080 # service port

targetPort: 8080 # container port

--> create load balancer

--> push the code

--> cd ../terraform-aws-eks/60-acm/

--> terraform init -reconfigure

**eks-argocd/applications/roboshop.yaml**

apiVersion: argoproj.io/v1alpha1

kind: Application

metadata:

name: roboshop

namespace: argocd # Namespace where the Argo CD Application resource lives

spec:

project: default # The Argo CD Project this application belongs to

source:

repoURL: https://github.com/daws-84s/eks-argocd.git # URL of the Git repository

targetRevision: main # The specific branch, tag, or commit to sync from

path: roboshop # The path within the repository where the application manifests are located

destination:

server: https://kubernetes.default.svc # The API server URL of the target Kubernetes cluster

namespace: argocd # The namespace in the target cluster where the application will be deployed

syncPolicy:

automated: # Enable automated synchronization

prune: true # Delete resources that are no longer in Git

selfHeal: true # Revert changes made manually in the cluster to match Git

syncOptions:

- CreateNamespace=true # Automatically create the target namespace if it doesn't exist

--> Amazon ECR

--> private registry