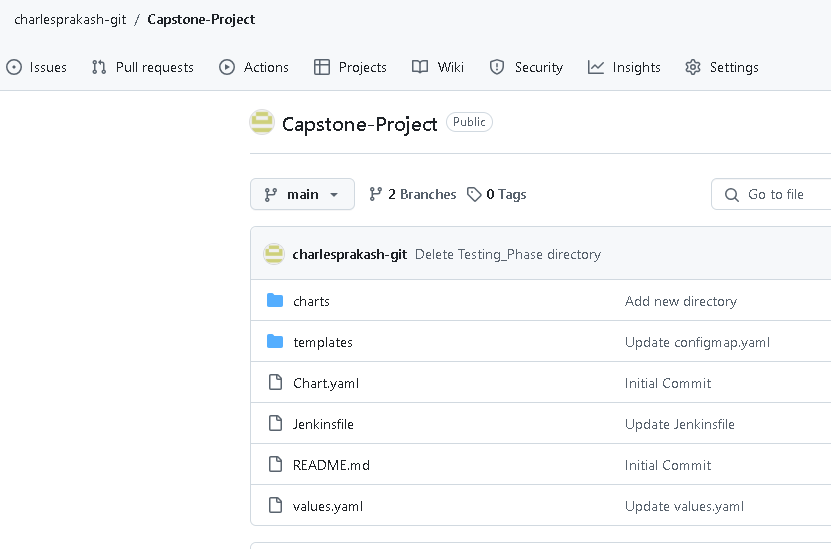
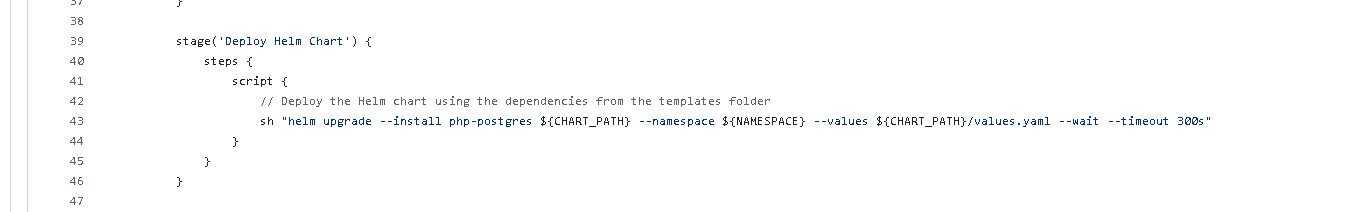
**Helm Package and Jenkins CI for Kubernetes Cluster**

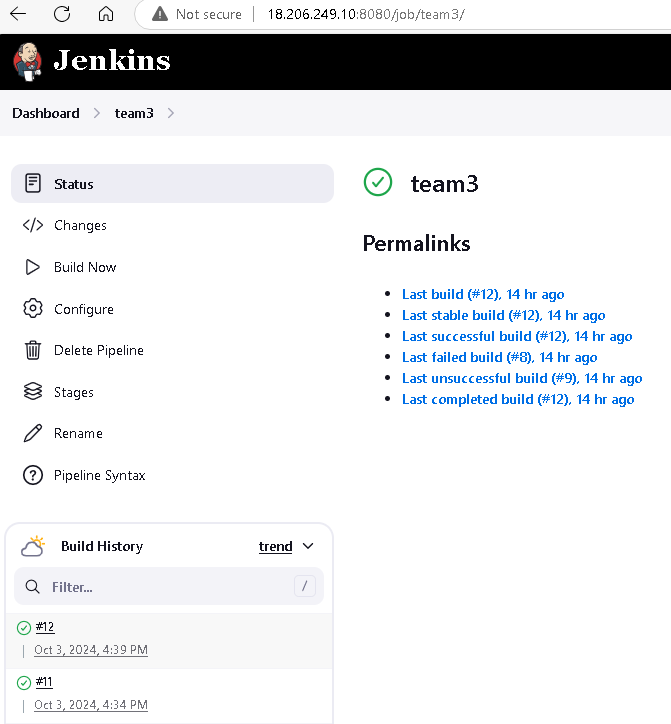
The successfully proven deployment code is available in SCM (GIT)  
  
URL:   
Staging & main (Production) Branches  
<https://github.com/charlesprakash-git/Capstone-Project.git>



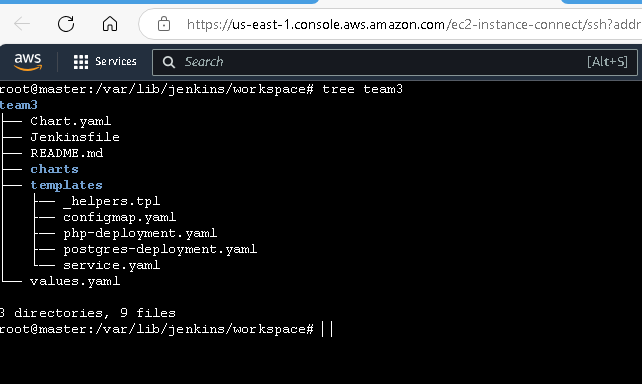
Deployment of the code is carried out by Jenkins Pipeline and Helm chart installation.  
  
**Jenkins deployment code through Helm**

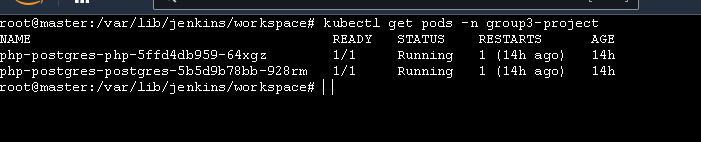


**Deployment pipeline in Jenkins**

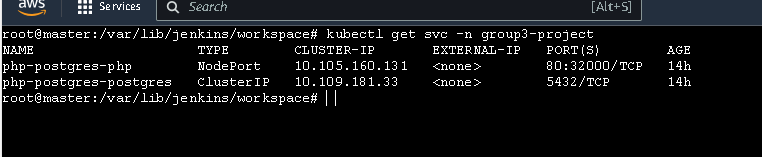


The following is the structure of the directory from Jenkins workspace

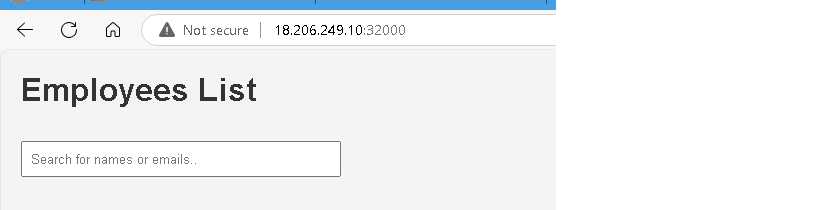


Kubernetes pods are successfully created in Kubernetes Master servers as below:  
  


Services are created successfully and exposed through Nodeport



End user application



**Following are the coding and Chart Directory Structure**

You will need to update the following files as part of Helm packaging.

**1. Chart.yaml**

yaml

Copy code

apiVersion: v2

name: php-postgres

description: A Helm chart to deploy PHP and PostgreSQL.

version: 1.0.0

appVersion: "1.0"

**2. values.yaml**

This file contains the default values for your chart. Users can override these values by providing their own values.yaml.

# Default values for php-postgres.

# PHP App Configuration

php:

image: php:8.1-apache

replicaCount: 1

servicePort: 8082 # Service port for PHP

dbHost: "postgres-service"

dbUser: "myuser"

dbPassword: "mypassword"

dbName: "mydb"

pvc:

size: 1Gi

service:

type: NodePort # Service type for PHP

# PostgreSQL Database Configuration

postgres:

image: postgres:15

replicaCount: 1

dbName: "mydb"

dbUser: "myuser"

dbPassword: "mypassword"

pvc:

size: 1Gi

# Remove the nodePort field from here as it is not needed

# If you want to access PostgreSQL externally, consider adding a service entry instead.

# PersistentVolumeClaims for both PHP and PostgreSQL

persistence:

enabled: true

storageClass: "efs-sc"

**3. templates/php-deployment.yaml**

This file defines the PHP deployment.

apiVersion: apps/v1

kind: Deployment

metadata:

name: {{ .Release.Name }}-php

labels:

app: php

spec:

replicas: {{ .Values.php.replicaCount }}

selector:

matchLabels:

app: php

template:

metadata:

labels:

app: php

spec:

containers:

- name: php

image: {{ .Values.php.image }}

ports:

- containerPort: {{ .Values.php.servicePort }}

env:

- name: DB\_HOST

value: {{ .Values.php.dbHost }}

- name: DB\_USER

value: {{ .Values.php.dbUser }}

- name: DB\_PASSWORD

value: {{ .Values.php.dbPassword }}

- name: DB\_NAME

value: {{ .Values.php.dbName }}

volumeMounts:

- name: php-persistent-storage

mountPath: /var/www/html

- name: php-configmap

mountPath: /var/www/html/index.php

subPath: index.php # Mounts only the index.php file from the ConfigMap

volumes:

- name: php-persistent-storage

persistentVolumeClaim:

claimName: {{ .Release.Name }}-php-pvc

- name: php-configmap

configMap:

name: {{ .Release.Name }}-init-sql-configmap # Name of the ConfigMap containing index.php

**4. templates/postgres-deployment.yaml**

This file defines the PostgreSQL deployment.

apiVersion: apps/v1

kind: Deployment

metadata:

name: {{ .Release.Name }}-postgres

labels:

app: postgres

spec:

replicas: {{ .Values.postgres.replicaCount }}

selector:

matchLabels:

app: postgres

template:

metadata:

labels:

app: postgres

spec:

containers:

- name: postgres

image: {{ .Values.postgres.image }}

ports:

- containerPort: 5432

env:

- name: POSTGRES\_DB

value: {{ .Values.postgres.dbName }}

- name: POSTGRES\_USER

value: {{ .Values.postgres.dbUser }}

- name: POSTGRES\_PASSWORD

value: {{ .Values.postgres.dbPassword }}

volumeMounts:

- name: postgres-data

mountPath: /var/lib/postgresql/data

- name: init-sql

mountPath: /docker-entrypoint-initdb.d

volumes:

- name: postgres-data

persistentVolumeClaim:

claimName: {{ .Release.Name }}-postgres-pvc

- name: init-sql

configMap:

name: {{ .Release.Name }}-init-sql-configmap

**5. templates/service.yaml**

This file defines the services for both the PHP and PostgreSQL applications.

apiVersion: v1

kind: Service

metadata:

name: {{ .Release.Name }}-php

spec:

type: NodePort

ports:

- port: {{ .Values.php.servicePort }}

targetPort: {{ .Values.php.servicePort }}

nodePort: {{ .Values.php.nodePort | default 30002 }} # Specify NodePort

selector:

app: php

---

apiVersion: v1

kind: Service

metadata:

name: {{ .Release.Name }}-postgres

spec:

type: ClusterIP

ports:

- port: 5432

targetPort: 5432

selector:

app: postgres

**6. templates/configmap.yaml**

This file defines the ConfigMap for initializing the PostgreSQL database.

apiVersion: v1

kind: ConfigMap

metadata:

name: {{ .Release.Name }}-init-sql-configmap

data:

init.sql: |

-- Create the employees table

CREATE TABLE employees (

employee\_id SERIAL PRIMARY KEY,

first\_name VARCHAR(100),

last\_name VARCHAR(100),

department VARCHAR(100),

job\_title VARCHAR(100),

email VARCHAR(100),

phone\_number VARCHAR(15),

hire\_date DATE,

status VARCHAR(20)

);

-- Insert 10 sample employees

INSERT INTO employees (first\_name, last\_name, department, job\_title, email, phone\_number, hire\_date, status) VALUES

('John', 'Doe', 'Sales', 'Sales Manager', 'john.doe@example.com', '(555) 123-4567', '2020-01-15', 'Active'),

('Jane', 'Smith', 'Marketing', 'Marketing Specialist', 'jane.smith@example.com', '(555) 234-5678', '2021-03-20', 'Active'),

('Alice', 'Johnson', 'IT', 'Software Engineer', 'alice.johnson@example.com', '(555) 345-6789', '2019-07-11', 'Inactive'),

('Bob', 'Brown', 'HR', 'HR Manager', 'bob.brown@example.com', '(555) 456-7890', '2018-11-05', 'Active'),

('Charlie', 'Davis', 'Finance', 'Accountant', 'charlie.davis@example.com', '(555) 567-8901', '2022-02-14', 'Active'),

('Diana', 'Moore', 'Sales', 'Sales Associate', 'diana.moore@example.com', '(555) 678-9012', '2023-05-01', 'Active'),

('Ethan', 'Clark', 'Marketing', 'SEO Specialist', 'ethan.clark@example.com', '(555) 789-0123', '2023-06-10', 'Active'),

('Fiona', 'Adams', 'IT', 'Systems Analyst', 'fiona.adams@example.com', '(555) 890-1234', '2022-09-15', 'Active'),

('George', 'Wright', 'Finance', 'Financial Analyst', 'george.wright@example.com', '(555) 901-2345', '2020-04-22', 'Inactive'),

('Hannah', 'Baker', 'HR', 'Recruiter', 'hannah.baker@example.com', '(555) 012-3456', '2021-08-30', 'Active');

index.php: |

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Employees List</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 20px;

padding: 0;

background-color: #f4f4f4;

}

h1 {

color: #333;

}

table {

width: 100%;

border-collapse: collapse;

margin: 20px 0;

}

table, th, td {

border: 1px solid #ddd;

}

th, td {

padding: 12px;

text-align: left;

}

th {

background-color: #f2f2f2;

}

input[type="text"] {

padding: 8px;

width: 300px;

margin: 10px 0;

}

</style>

<script>

// Search Functionality

function searchUsers() {

var input, filter, table, tr, td, i, txtValue;

input = document.getElementById("searchInput");

filter = input.value.toUpperCase();

table = document.getElementById("employeesTable");

tr = table.getElementsByTagName("tr");

for (i = 1; i < tr.length; i++) {

tr[i].style.display = "none";

td = tr[i].getElementsByTagName("td");

for (var j = 0; j < td.length; j++) {

if (td[j]) {

txtValue = td[j].textContent || td[j].innerText;

if (txtValue.toUpperCase().indexOf(filter) > -1) {

tr[i].style.display = "";

break;

}

}

}

}

}

</script>

</head>

<body>

<h1>Employees List</h1>

<!-- Search Box -->

<input type="text" id="searchInput" onkeyup="searchUsers()" placeholder="Search for names or emails..">

<?php

// Connection details

$host = getenv('DB\_HOST');

$dbname = getenv('DB\_NAME');

$user = getenv('DB\_USER');

$password = getenv('DB\_PASSWORD');

$dsn = "pgsql:host=$host;dbname=$dbname";

try {

// Connect to PostgreSQL database

$pdo = new PDO($dsn, $user, $password, [PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION]);

// Fetch data from the employees table

$stmt = $pdo->query("SELECT \* FROM employees");

$employees = $stmt->fetchAll(PDO::FETCH\_ASSOC);

// Start of table

echo "<table id='employeesTable'>";

echo "<tr><th>ID</th><th>First Name</th><th>Last Name</th><th>Department</th><th>Job Title</th><th>Email</th><th>Phone</th><th>Hire Date</th><th>Status</th></tr>";

// Display the employees in the table

foreach ($employees as $employee) {

echo "<tr>

<td>{$employee['employee\_id']}</td>

<td>{$employee['first\_name']}</td>

<td>{$employee['last\_name']}</td>

<td>{$employee['department']}</td>

<td>{$employee['job\_title']}</td>

<td>{$employee['email']}</td>

<td>{$employee['phone\_number']}</td>

<td>{$employee['hire\_date']}</td>

<td>{$employee['status']}</td>

</tr>";

}

echo "</table>";

} catch (PDOException $e) {

echo "Error: " . $e->getMessage();

}

?>

</body>

</html>

**7. templates/pvc.yaml**

This file defines the persistent volume claims for both the PHP and PostgreSQL applications.

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: {{ .Release.Name }}-php-pvc

spec:

storageClassName: {{ .Values.persistence.storageClass | quote }} # This adds the storage class from values.yaml

accessModes:

- ReadWriteOnce

resources:

requests:

storage: {{ .Values.php.pvc.size }}

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: {{ .Release.Name }}-postgres-pvc

spec:

storageClassName: {{ .Values.persistence.storageClass | quote }} # This adds the storage class from values.yaml

accessModes:

- ReadWriteOnce

resources:

requests:

storage: {{ .Values.postgres.pvc.size }}

**8. \_helpers.tpl**

This is used for helper templates such as label selectors or annotations. It is not required for basic setups but can be added for future customizations.

**Install the Helm Chart**

To package and install the Helm chart, follow these steps:

1. **Package the Helm chart**:

bash

Copy code

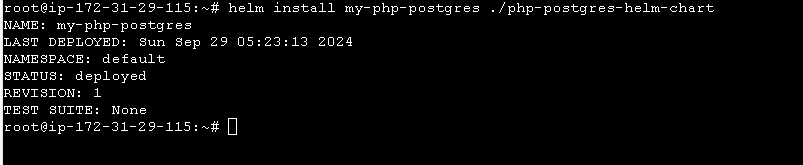
helm package ./php-postgres-helm-chart

1. **Install the chart**:

bash

Copy code

helm install my-php-postgres ./php-postgres-helm-chart



1. **Check the deployment**:

bash

Copy code

kubectl get pods

kubectl get services

This Helm chart encapsulates the full deployment for both PHP and PostgreSQL, along with persistent volume claims and database

Automating the Helm package deployment through Jenkins involves creating a Jenkins pipeline that builds the Helm chart, packages it, and then deploys it to a Kubernetes cluster. Below are the steps to set up Jenkins for automating this process:

**Create a Jenkins Pipeline Job**

1. **Create a new item in Jenkins**:
   * Go to the Jenkins dashboard.
   * Click on “New Item” and select “Pipeline”.
2. **Configure the pipeline**:
   * In the pipeline configuration, set the pipeline script to use a Jenkinsfile (or you can directly write the pipeline script in the script box).

**Create a Jenkinsfile**

Here’s the Jenkinsfile that automates the Helm packaging and deployment process:

Copy code

pipeline {

agent any

environment {

KUBE\_CONFIG = credentials('kubeconfig') // Kubernetes configuration

HELM\_HOME = "${env.WORKSPACE}/.helm" // Helm home directory

}

stages {

stage('Clone Repository') {

steps {

git 'https://your-repo-url.git' // Clone your repository

}

}

stage('Install Helm') {

steps {

sh 'curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash' // Install Helm

}

}

stage('Package Helm Chart') {

steps {

dir('php-postgres-helm-chart') { // Navigate to your chart directory

sh 'helm package .'

}

}

}

stage('Deploy to Kubernetes') {

steps {

script {

// Setting the KUBECONFIG variable for Helm to find your cluster config

withCredentials([file(credentialsId: 'kubeconfig', variable: 'KUBE\_CONFIG')]) {

sh 'export KUBECONFIG=$KUBE\_CONFIG'

sh 'helm upgrade --install my-php-postgres ./php-postgres-helm-chart --values ./php-postgres-helm-chart/values.yaml'

}

}

}

}

}

post {

success {

echo 'Helm chart deployed successfully!'

}

failure {

echo 'Helm chart deployment failed.'

}

}

}

**Configure Credentials**

1. **Git Credentials** (if necessary):
   * If your Git repository requires authentication, add the Git credentials as well.

**Monitor the Pipeline**

* As the pipeline runs, you can monitor the progress and logs from the Jenkins dashboard. It will output logs for each stage, showing you the success or failure of the packaging and deployment steps.

**Conclusion**

By setting up this Jenkins pipeline, you can automate the Helm packaging and deployment process, allowing for continuous integration and continuous deployment (CI/CD) of your PHP and PostgreSQL applications to Kubernetes. This automation improves efficiency and reduces the risk of human error during deployments.