# Kubernetes Flask-MySQL Project

## Introduction

This repository contains a comprehensive project demonstrating a robust CI/CD pipeline for a Flask application integrated with a MySQL database, all orchestrated within a Kubernetes environment. The project emphasizes best practices in containerization, automated testing, and continuous deployment using GitHub Actions and ArgoCD. This README provides a detailed guide to setting up, deploying, and managing the application, along with insights into its architecture and testing methodologies.

## Table of Contents

* [1. Project Objective](#1-project-objective)
* [2. Setup](#2-setup)
  + [2.1. Virtual Machine Setup (VMware Workstation)](#21-virtual-machine-setup-vmware-workstation)
  + [2.2. K3s Installation](#22-k3s-installation)
* [3. GitHub Repository Setup](#3-github-repository-setup)
* [4. Kubernetes YAML Files](#4-kubernetes-yaml-files)
  + [4.1. Flask Application Manifests](#41-flask-application-manifests)
  + [4.2. MySQL Database Manifests](#42-mysql-database-manifests)
  + [4.3. Kustomization](#43-kustomization)
* [5. Application Code](#5-application-code)
  + [5.1. Flask Application (app.py)](#51-flask-application-apppy)
  + [5.2. Database Initialization (init.sql)](#52-database-initialization-initsql)
* [6. Testing](#6-testing)
  + [6.1. Unit Tests (Pytest)](#61-unit-tests-pytest)
  + [6.2. Linting](#62-linting)
  + [6.3. Coverage Report](#63-coverage-report)
* [7. Docker Image Build and Push](#7-docker-image-build-and-push)
  + [7.1. Dockerfile for Flask](#71-dockerfile-for-flask)
  + [7.2. Dockerfile for DB](#72-dockerfile-for-db)
* [8. CI/CD Pipeline (GitHub Actions)](#8-cicd-pipeline-github-actions)
* [9. Continuous Deployment (ArgoCD)](#9-continuous-deployment-argocd)
  + [9.1. ArgoCD Installation and Configuration](#91-argocd-installation-and-configuration)
* [10. Directory Structure](#10-directory-structure)
* [11. Validation & Testing](#11-validation--testing)
* [12. Submission Checklist](#12-submission-checklist)
* [Conclusion](#conclusion)

## 1. Project Objective

This project aims to build a complete CI/CD workflow for a Flask application backed by MySQL. The application will be containerized, tested, and deployed to Kubernetes using Kustomize with separate overlays for testing and production environments.

## 2. Setup

### 2.1. Virtual Machine Setup (VMware Workstation)

To set up the working environment, you will need two Virtual Machines (VMs) running CentOS 7 or 8 on VMware Workstation:

* **k8s-master**: Master node for the Kubernetes cluster.
* **k8s-worker**: Worker node for the Kubernetes cluster.
* **NFS**

**VM Specifications for each VM:**

| **Component** | **Value** |
| --- | --- |
| RAM | 2 GB or more |
| CPU | 2 Core |
| Disk | 20 GB |
| Network | Bridged or Host-only |
| SSH | Enabled |

### A screenshot of a computer AI-generated content may be incorrect.

### 2.2. K3s Installation

**On the Master VM:**

1. Execute the following command to install k3s:

curl -sfL https://get.k3s.io | sh -

1. After installation, retrieve the node token (needed for worker node joining):

sudo cat /var/lib/rancher/k3s/server/node-token

1. Note down the Master VM's IP address (e.g., 192.168.2.154).

**On the Worker VM:**

1. Use the Master's IP and token to join the cluster:

curl -sfL https://get.k3s.io | K3S\_URL=https://<MASTER\_IP>:6443 K3S\_TOKEN=<TOKEN> sh -

Replace <MASTER\_IP> with your Master VM's IP and <TOKEN> with the token obtained from the Master.

**Verify Cluster Status (On Master VM):**

sudo k3s kubectl get nodes

Ensure both master and worker nodes show Ready status.

## 3. GitHub Repository Setup

Create a new repository on GitHub and initialize Git in your VM machine.

## 4. Kubernetes YAML Files

This section outlines the Kubernetes manifest files used for deploying the Flask application and MySQL database.

### 4.1. Flask Application Manifests

**flask-configmap.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: flask-config

data:

MYSQL\_DATABASE\_HOST: db-service

MYSQL\_DATABASE\_DB: BucketList

MYSQL\_DATABASE\_USER: flaskuser

**flask-deployment.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: flask-app

spec:

replicas: 2

selector:

matchLabels:

app: flask

template:

metadata:

labels:

app: flask

spec:

containers:

- name: flask-app

image: tarekadel/k8s-flask-mysql-flaskapp:v10

ports:

- containerPort: 80

resources:

requests:

memory: "64Mi"

cpu: "100m"

limits:

memory: "128Mi"

cpu: "250m"

env:

- name: MYSQL\_DATABASE\_USER

valueFrom:

configMapKeyRef:

name: flask-config

key: MYSQL\_DATABASE\_USER

- name: MYSQL\_DATABASE\_PASSWORD

valueFrom:

secretKeyRef:

name: mysql-secret

key: mysql-password

- name: MYSQL\_DATABASE\_DB

valueFrom:

configMapKeyRef:

name: flask-config

key: MYSQL\_DATABASE\_DB

- name: MYSQL\_DATABASE\_HOST

valueFrom:

configMapKeyRef:

name: flask-config

key: MYSQL\_DATABASE\_HOST

startupProbe:

httpGet:

path: /healthz

port: 80

initialDelaySeconds: 5

periodSeconds: 5

failureThreshold: 30

readinessProbe:

httpGet:

path: /healthz

port: 80

initialDelaySeconds: 10

periodSeconds: 10

failureThreshold: 3

livenessProbe:

httpGet:

path: /healthz

port: 80

initialDelaySeconds: 20

periodSeconds: 20

failureThreshold: 3

**flask-ingress.yaml**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: flask-ingress

namespace: default

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

ingressClassName: nginx

rules:

- http:

paths:

- path: /flask

pathType: Prefix

backend:

service:

name: flask-service

port:

number: 80

**flask-nodeport.yaml**

apiVersion: v1

kind: Service

metadata:

name: flask-nodeport

spec:

type: NodePort

selector:

app: flask

ports:

- port: 80

targetPort: 80

nodePort: 30080

**flask-service.yaml**

apiVersion: v1

kind: Service

metadata:

name: flask-service

spec:

selector:

app: flask

ports:

- protocol: TCP

port: 80

targetPort: 80

### 4.2. MySQL Database Manifests

**limit-range.yaml**

apiVersion: v1

kind: LimitRange

metadata:

name: default-limits

spec:

limits:

- default:

memory: 1Gi

cpu: 500m

defaultRequest:

memory: 512Mi

cpu: 300m

type: Container

**mysql-configmap.yaml**

apiVersion: v1

kind: ConfigMap

metadata:

name: mysql-config

labels:

app: mysql

data:

MYSQL\_DATABASE: flaskdb

**mysql-network-policy.yaml**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: mysql-network-policy

spec:

podSelector:

matchLabels:

app: mysql

ingress:

- from:

- podSelector:

matchLabels:

app: flask

- podSelector:

matchLabels:

app: debug

ports:

- protocol: TCP

port: 3306

policyTypes:

- Ingress

**mysql-secret.yaml**

apiVersion: v1

kind: Secret

metadata:

name: mysql-secret

labels:

app: mysql

type: Opaque

data:

root-password: cm9vdA== # "root"

mysql-user: Zmxhc2t1c2Vy # "flaskuser"

mysql-password: eW91cnBhc3N3b3Jk # "yourpassword"

**mysql-service.yaml**

apiVersion: v1

kind: Service

metadata:

name: db-service

spec:

selector:

app: mysql

ports:

- port: 3306

targetPort: 3306

type: ClusterIP

**mysql-statefulset.yaml**

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: mysql

spec:

serviceName: db-service

replicas: 2

selector:

matchLabels:

app: mysql

template:

metadata:

labels:

app: mysql

spec:

containers:

- name: mysql

image: tarekadel/k8s-flask-mysql-database:v13

args: ["--port=3306"]

ports:

- containerPort: 3306

env:

- name: MYSQL\_ROOT\_PASSWORD

valueFrom:

secretKeyRef:

name: mysql-secret

key: root-password

- name: MYSQL\_DATABASE

valueFrom:

configMapKeyRef:

name: mysql-config

key: MYSQL\_DATABASE

- name: MYSQL\_USER

valueFrom:

secretKeyRef:

name: mysql-secret

key: mysql-user

- name: MYSQL\_PASSWORD

valueFrom:

secretKeyRef:

name: mysql-secret

key: mysql-password

volumeMounts:

- name: mysql-persistent-storage

mountPath: /var/lib/mysql

imagePullSecrets:

- name: regcred

volumeClaimTemplates:

- metadata:

name: mysql-persistent-storage

spec:

accessModes: ["ReadWriteMany"]

storageClassName: nfs-storage

resources:

requests:

storage: 1Gi

**nfs-provisioner-deployment.yaml**

apiVersion: v1

kind: ServiceAccount

metadata:

name: nfs-client-provisioner

namespace: nfs-provisioner

---

kind: ClusterRole

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: nfs-client-provisioner-runner

rules:

- apiGroups: [""]

resources: ["persistentvolumes"]

verbs: ["get", "list", "watch", "create", "delete"]

- apiGroups: [""]

resources: ["persistentvolumeclaims"]

verbs: ["get", "list", "watch", "update"]

- apiGroups: ["storage.k8s.io"]

resources: ["storageclasses"]

verbs: ["get", "list", "watch"]

- apiGroups: [""]

resources: ["events"]

verbs: ["create", "update", "patch"]

- apiGroups: [""]

resources: ["endpoints"]

verbs: ["get", "list", "watch", "create", "update", "patch"]

---

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: run-nfs-client-provisioner

subjects:

- kind: ServiceAccount

name: nfs-client-provisioner

namespace: nfs-provisioner

roleRef:

kind: ClusterRole

name: nfs-client-provisioner-runner

apiGroup: rbac.authorization.k8s.io

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: nfs-client-provisioner

namespace: nfs-provisioner

spec:

replicas: 1

selector:

matchLabels:

app: nfs-client-provisioner

template:

metadata:

labels:

app: nfs-client-provisioner

spec:

serviceAccountName: nfs-client-provisioner

containers:

- name: nfs-client-provisioner

image: registry.k8s.io/sig-storage/nfs-subdir-external-provisioner:v4.0.2

volumeMounts:

- name: nfs-client-root

mountPath: /persistentvolumes

env:

- name: PROVISIONER\_NAME

value: nfs-client

- name: NFS\_SERVER

value: 192.168.2.155 # <-- Update with your NFS server IP

- name: NFS\_PATH

value: /mnt/nfs-share

volumes:

- name: nfs-client-root

nfs:

server: 192.168.2.155 # <-- Same IP as above

path: /mnt/nfs-share

**nfs-storageclass.yaml**

# nfs-storageclass.yaml

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: nfs-storage

provisioner: nfs-client

parameters:

archiveOnDelete: "false"

reclaimPolicy: Retain

volumeBindingMode: Immediate

**resource-quota.yaml**

apiVersion: v1

kind: ResourceQuota

metadata:

name: app-quota

spec:

hard:

requests.cpu: "4"

requests.memory: 8Gi

limits.cpu: "4"

limits.memory: 8Gi

pods: "10"

persistentvolumeclaims: "5"

### 4.3. Kustomization

kustomization.yaml will contain all required YAML files to be configured in overlays for both test and production environments with different versions. These files include:

* flask-configmap-prod.yaml
* flask-deployment-prod.yaml
* flask-ingress-prod.yaml
* flask-nodeport-prod.yaml
* flask-service-prod.yaml
* mysql-configmap-prod.yaml
* mysql-network-policy-prod.yaml
* mysql-secret-prod.yaml
* mysql-service-prod.yaml
* mysql-statefulset-prod.yaml

**A screen shot of a computer

AI-generated content may be incorrect.**

Configuration differences based on environment:

| **Feature** | **Testing** | **Production** |
| --- | --- | --- |
| Replicas | 1 | 3 |
| CPU Limit | 200m | 500m |
| Memory | 256Mi | 1024Mi |

## 5. Application Code

### 5.1. Flask Application (app.py)

from flask import Flask, request, render\_template\_string

import pymysql

app = Flask(\_\_name\_\_)

def get\_connection():

return pymysql.connect(

host="db-service",

port=3306,

user="flaskuser",

password="yourpassword",

database="flaskdb",

cursorclass=pymysql.cursors.DictCursor

)

@app.route('/healthz')

def healthz():

return "OK", 200

@app.route('/')

def home():

try:

conn = get\_connection()

cursor = conn.cursor()

cursor.execute("SELECT count FROM counter WHERE id=1;")

result = cursor.fetchone()

count = result['count'] if result else 0

conn.close()

html = f'''

<h1>Counter: {count}</h1>

<form method="POST" action="/increment">

<button type="submit">Increment</button>

</form>

'''

return html

except Exception as e:

return f"Error: {e}"

@app.route('/flask')

def flask\_route():

return home()

@app.route('/increment', methods=['POST'])

def increment():

try:

conn = get\_connection()

cursor = conn.cursor()

cursor.execute("UPDATE counter SET count = count + 1 WHERE id = 1;")

conn.commit()

conn.close()

return '<script>window.location.href = "/";</script>'

except Exception as e:

return f"Error: {e}"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=80)

### 5.2. Database Initialization (init.sql)

CREATE DATABASE IF NOT EXISTS flaskdb;

CREATE USER IF NOT EXISTS 'flaskuser'@'%' IDENTIFIED WITH mysql\_native\_password BY 'yourpassword';

GRANT ALL PRIVILEGES ON flaskdb.\* TO 'flaskuser'@'%';

FLUSH PRIVILEGES;

USE flaskdb;

CREATE TABLE IF NOT EXISTS users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) NOT NULL UNIQUE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE IF NOT EXISTS counter (

id INT PRIMARY KEY,

count INT NOT NULL DEFAULT 0

);

INSERT INTO counter (id, count)

SELECT 1, 0

WHERE NOT EXISTS (SELECT \* FROM counter WHERE id = 1);

## 6. Testing

### 6.1. Unit Tests (Pytest)

Unit tests for the Flask application are located in flask/tests/test\_app.py and cover routes, DB interaction simulation, and application behavior.

**test\_app.py**

import pytest

from unittest.mock import patch, MagicMock

from k8s.base.flaskapp.app import app

@pytest.fixture

def client():

with app.test\_client() as client:

yield client

def test\_healthz(client):

resp = client.get('/healthz')

assert resp.status\_code == 200

assert resp.data == b"OK"

@patch('k8s.base.flaskapp.app.get\_connection')

def test\_home\_route(mock\_get\_conn, client):

mock\_cursor = MagicMock()

mock\_cursor.fetchone.return\_value = {'count': 5}

mock\_conn = MagicMock()

mock\_conn.cursor.return\_value = mock\_cursor

mock\_get\_conn.return\_value = mock\_conn

response = client.get('/')

assert response.status\_code == 200

assert b"Counter: 5" in response.data

@patch('k8s.base.flaskapp.app.get\_connection')

def test\_flask\_route(mock\_get\_conn, client):

mock\_cursor = MagicMock()

mock\_cursor.fetchone.return\_value = {'count': 7}

mock\_conn = MagicMock()

mock\_conn.cursor.return\_value = mock\_cursor

mock\_get\_conn.return\_value = mock\_conn

response = client.get('/flask')

assert response.status\_code == 200

assert b"Counter: 7" in response.data

@patch('k8s.base.flaskapp.app.get\_connection')

def test\_increment\_route(mock\_get\_conn, client):

mock\_cursor = MagicMock()

mock\_conn = MagicMock()

mock\_conn.cursor.return\_value = mock\_cursor

mock\_get\_conn.return\_value = mock\_conn

response = client.post('/increment')

assert response.status\_code == 200

assert b"window.location.href" in response.data

mock\_cursor.execute.assert\_called\_with("UPDATE counter SET count = count + 1 WHERE id = 1;")

mock\_conn.commit.assert\_called\_once()

**requirements-tests.txt**

Flask==2.3.3

PyMySQL==1.1.0

cryptography==42.0.5

pytest

pytest-mock

coverage

flake8

sqlfluff

### 6.2. Linting

* **Flake8 (Flask App)**: Checks for formatting, potential issues, line length, and other organizational improvements to enhance code quality.
* **SQLFluff (DB App)**: Checks for formatting, potential issues, line length, and other organizational improvements to enhance code quality.

### 6.3. Coverage Report

Generate a coverage report using coverage.py with a target of ≥ 80%.

## 7. Docker Image Build and Push

Docker images will be built for both the Flask application and MySQL database. These images will then be pushed to a Docker registry.

**Commit Message Format for Image Builds:**

[build-image] flask:v11 db:v14 (example)

### 7.1. Dockerfile for Flask

# Use an official Python base image

FROM python:3.10-slim

# Set the working directory inside the container

WORKDIR /app

# Copy requirements file and install dependencies

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

# Copy the Flask application code

COPY app.py .

# Expose the Flask port

EXPOSE 80

# Command to run the app

CMD ["python", "app.py"]

### 7.2. Dockerfile for DB

FROM mysql:8.0

COPY init.sql /docker-entrypoint-initdb.d/

COPY my.cnf /etc/mysql/my.cnf

COPY my.cnf /etc/my.cnf

## 8. CI/CD Pipeline (GitHub Actions)

The CI/CD pipeline uses GitHub Actions to automate testing, building, and deployment.

**ci.yml (Example Workflow)**

name: Run Pytest and Lint

on:

push:

branches: [main]

pull\_request:

branches: [main]

jobs:

test:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v3

with:

token: ${{ secrets.GITTHUB\_TOKEN }}

- name: Set up Python "3.10"

uses: actions/setup-python@v4

with:

python-version: "3.10"

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r k8s/base/flaskapp/tests/requirements-tests.txt

- name: Lint with flake8 and reviewdog

uses: reviewdog/action-flake8@v3

with:

# ... (additional flake8 configuration)

## 9. Continuous Deployment (ArgoCD)

ArgoCD will monitor the GitHub repository for changes in the Kubernetes manifest files. Upon detecting changes, ArgoCD will apply them to the local environment, reflecting updates managed by Kustomize.

### 9.1. ArgoCD Installation and Configuration

1. Create the ArgoCD namespace:

kubectl create namespace argocd

1. Apply the ArgoCD installation manifests:

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

1. Port-forward the ArgoCD server to access the UI:

kubectl port-forward svc/argocd-server -n argocd 8080:443

1. Open https://localhost:8080 in your web browser to configure ArgoCD. Remember to provide a GitHub token for read permissions.

**Dynamic Pull for Changes:**

ArgoCD will pull changes dynamically every 5 minutes if the runner changes tags in the files:

\*/5 \* \* \* \* cd /path/to/K8s-Flask-MySQL-Project && git pull origin main

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

## 10. Directory Structure

K8s-Flask-MySQL-Project/

├── database/ # MySQL database configuration

│ ├── Dockerfile # MySQL container configuration

│ ├── init.sql # Database initialization script

│ └── my.cnf # MySQL configuration file

├── flaskapp/ # Flask web application

│ ├── app.py # Main Flask application

│ ├── Dockerfile # Flask container configuration

│ └── requirements.txt # Python dependencies

├── k8s/ # Kubernetes manifests

│ ├── base/

│ │ ├── deployment.yaml

│ │ ├── service.yaml

│ │ ├── configmap.yaml

│ │ ├── pvc.yaml

│ │ ├── ingress.yaml

│ │ └── kustomization.yaml

│ └── overlays/

│ ├── testing/

│ └── production/

├── .github/ # GitHub Actions workflows

│ └── workflows/

│ └── ci.yml

├── flask/tests/ # Flask application tests

│ ├── test\_app.py

│ └── requirements-tests.txt

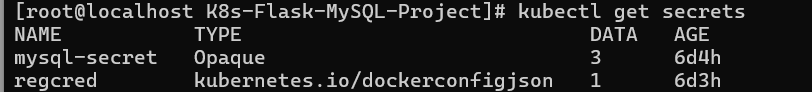
└── README.md # This file

**A screenshot of a computer program

AI-generated content may be incorrect.**

## 11. Validation & Testing

**Store DB credentials securely using secrets.**

****

**Validate FlaskApp ↔ MySQL database connectivity**

A screenshot of a computer

AI-generated content may be incorrect.

**Confirm ingress path /flask is accessible.**

* **A screenshot of a computer

  AI-generated content may be incorrect.**
* **Verify resource limits are correctly applied per environment.**
* **For flask:**
* **A computer screen with white text

  AI-generated content may be incorrect.**
* **For db**
* **A screen shot of a computer

  AI-generated content may be incorrect.**
* **Ensure secrets are injected and used properly.**
* **A black screen with white text

  AI-generated content may be incorrect.**
* **A computer screen with white text

  AI-generated content may be incorrect.**
* **Verify data persists in MySQL after restarts.**
* **A screen shot of a computer program

  AI-generated content may be incorrect.**

## Conclusion

This project provides a comprehensive example of building, testing, and deploying a containerized Flask application with a MySQL database on Kubernetes, leveraging a robust CI/CD pipeline. By following the steps outlined in this README, you can replicate the environment, understand the underlying technologies, and adapt the solution for your own projects. The use of Kustomize for environment-specific configurations and ArgoCD for continuous deployment ensures a scalable and maintainable infrastructure.

**Happy Deploying! 🚀**