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

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

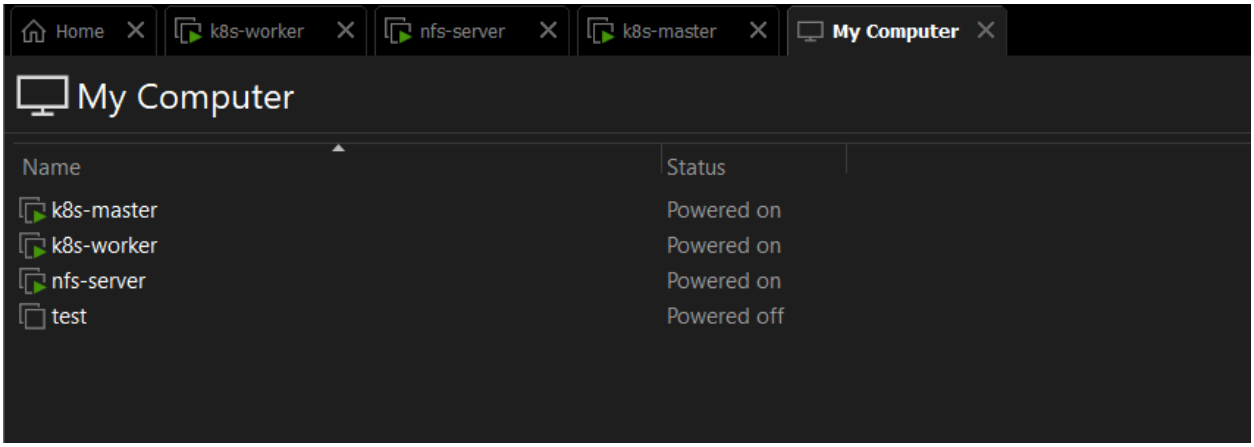
2.1. Virtual Machine Setup (VMware Workstation)

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



2.2. K3s Installation

On the Master VM:

1. Execute the following command to install k3s:

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

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

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

3. 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`

```
[root@localhost K8s-Flask-MySQL-Project]# cat k8s/overlays/production/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
  - 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

[root@localhost K8s-Flask-MySQL-Project]#
```

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
   ```

2. Apply the ArgoCD installation manifests:

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

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

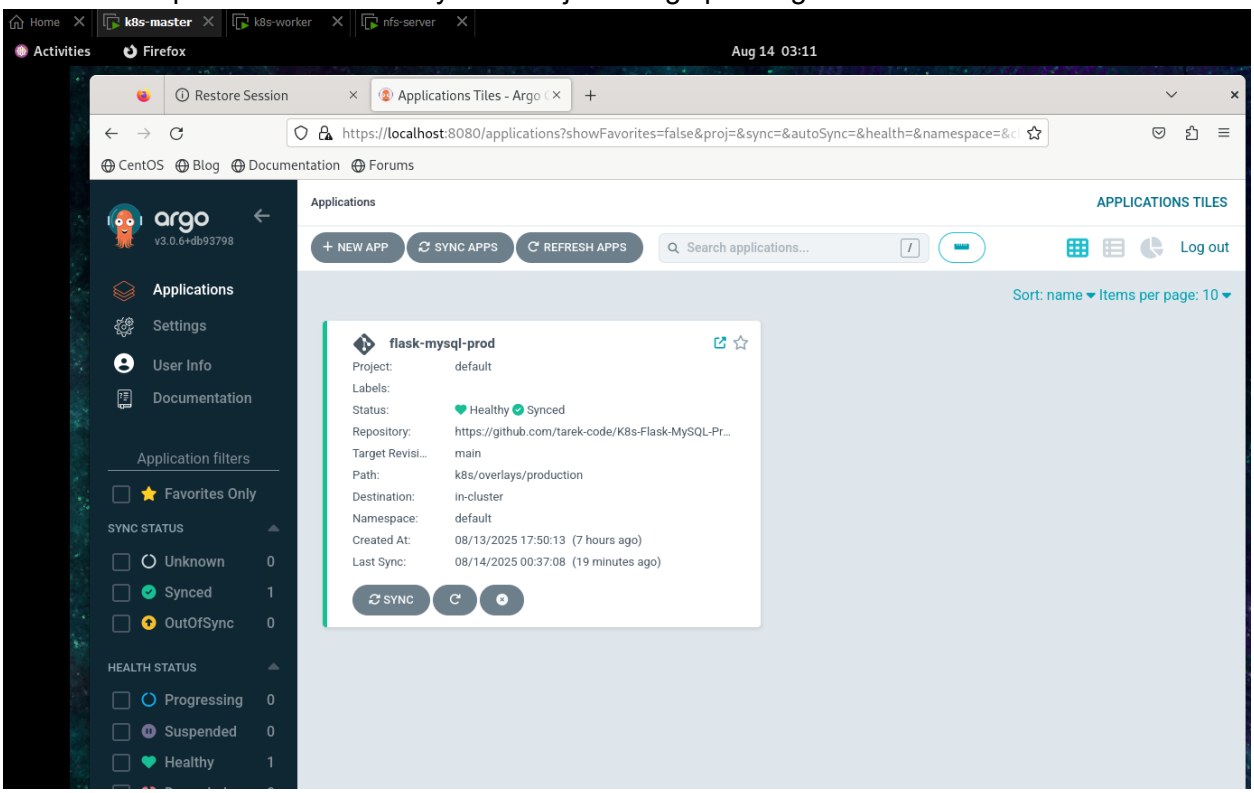
   kubectl port-forward svc/argocd-server -n argocd 8080:443

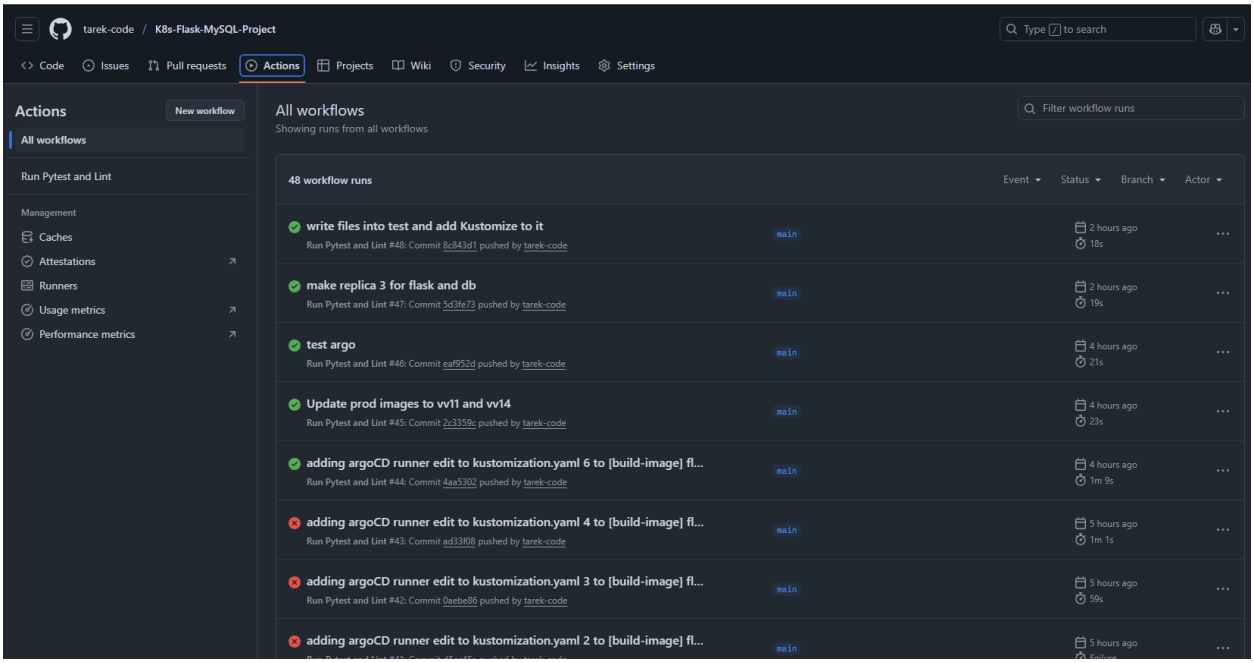
4. Open `https://localhost:8080` in your web browser to configure ArgoCD. Remember to provide a GitHub token for read permissions.

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





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

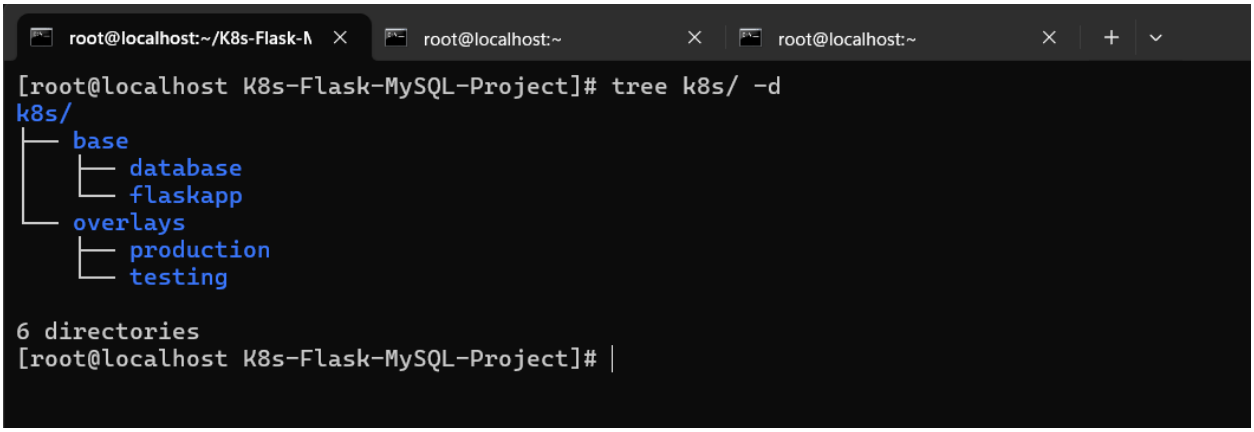
│ └── requirements.txt # Python dependencies

├── 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
```

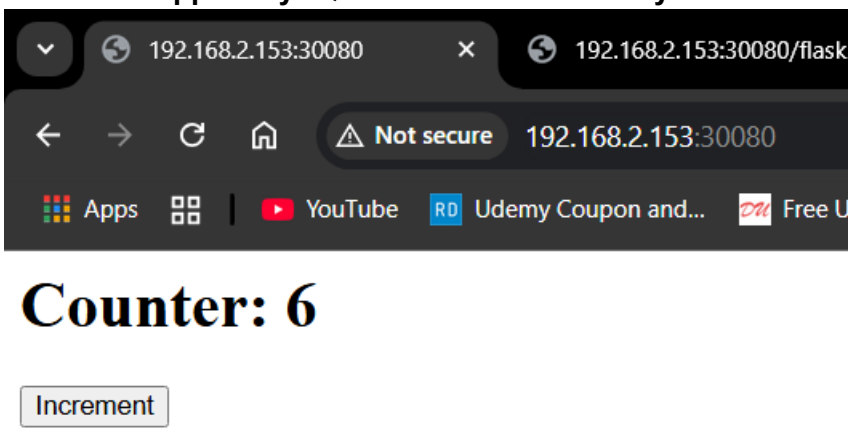


11. Validation & Testing

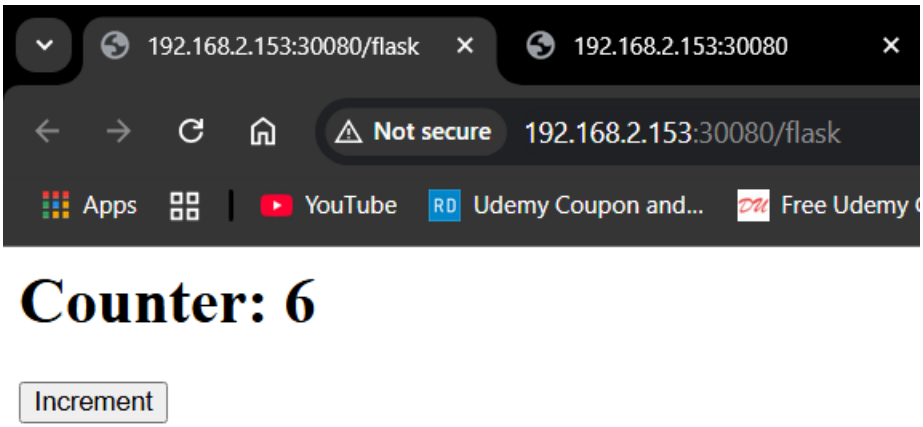
Store DB credentials securely using secrets.

```
[root@localhost K8s-Flask-MySQL-Project]# kubectl get secrets
NAME           TYPE                                  DATA   AGE
mysql-secret   Opaque                                3      6d4h
regcred        kubernetes.io/dockerconfigjson        1      6d3h
```

Validate FlaskApp ↔ MySQL database connectivity



Confirm ingress path `/flask` is accessible.



-
- **Verify resource limits are correctly applied per environment.**
- **For flask:**

```
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Thu, 14 Aug 2025 00:11:25 +0300
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     500m
      memory:  1Gi
    Requests:
      cpu:      100m
      memory:   64Mi
    Liveness:   http-get http://:80/healthz delay=20s timeout=1s period=20s #success=1 #failure=3
    Readiness:  http-get http://:80/healthz delay=10s timeout=1s period=10s #success=1 #failure=3
    Startup:    http-get http://:80/healthz delay=5s timeout=1s period=5s #success=1 #failure=30
```

-
- **For db**

```
    Limits:
      cpu:     500m
      memory:  1Gi
    Requests:
      cpu:     100m
      memory:  64Mi
    Environment:
      MYSQL_ROOT_PASSWORD:  <set to the key 'root-password' in secret 'mysql-secret'>       Optional: false
      MYSQL_DATABASE:       <set to the key 'MYSQL_DATABASE' of config map 'mysql-config'>  Optional: false
      MYSQL_USER:           <set to the key 'mysql-user' in secret 'mysql-secret'>          Optional: false
      MYSQL_PASSWORD:       <set to the key 'mysql-password' in secret 'mysql-secret'>      Optional: false
    Mounts:
```

-

- **Ensure secrets are injected and used properly.**

```
[root@localhost K8s-Flask-MySQL-Project]# kubectl exec -it mysql-0 -- sh
sh-5.1# echo $MYSQL_PASSWORD
yourpassword
sh-5.1# echo $MYSQL_USER
flaskuser
sh-5.1#
```

-
-

```
[root@localhost K8s-Flask-MySQL-Project]# kubectl get secrets
NAME           TYPE                             DATA   AGE
mysql-secret   Opaque                           3      6d4h
regcred        kubernetes.io/dockerconfigjson   1      6d4h
[root@localhost K8s-Flask-MySQL-Project]# kubectl describe secret regcred
Name:         regcred
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  kubernetes.io/dockerconfigjson

Data
====
.dockerconfigjson:  223 bytes
[root@localhost K8s-Flask-MySQL-Project]# kubectl describe secret mysql-secret
Name:         mysql-secret
Namespace:    default
Labels:       app=mysql
Annotations:  argocd.argoproj.io/tracking-id: flask-mysql-prod:/Secret:default/mysql-secret

Type:  Opaque

Data
====
mysql-password:  12 bytes
mysql-user:      9 bytes
root-password:   4 bytes
```

-
- **Verify data persists in MySQL after restarts.**

```
  volumeClaimTemplates:
  - metadata:
      name: mysql-persistent-storage
    spec:
      accessModes: ["ReadWriteMany"]
      storageClassName: nfs-storage
      resources:
        requests:
          storage: 1Gi
```

-

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! 🚀