Project Documentation: Wisecow Application Deployment with CI/CD and Kubernetes

Table of Contents

1. Project Overview

The Wisecow project demonstrates a full deployment pipeline from development to production using Docker, Kubernetes, and GitHub Actions for CI/CD. The goal was to containerize the application, deploy it on a Kubernetes cluster, and automate build and deployment processes.

2. Technologies Used

- **Docker**: For containerizing the application.
- **Docker Hub**: For hosting the container image.
- **Kubernetes**: For application deployment, scaling, and management.
- **GitHub Actions**: For implementing CI/CD pipelines.

3. Dockerizing the Application

The first step involved creating a Docker image for the application:

- A Dockerfile was created, specifying the base image and required dependencies.
- The application was copied into the Docker container and configured to run on a specified port (port 4499).
- Built the Docker image and tested it locally with the following commands:

docker build -t britneycorreia/wisecow-app:latest .

docker run -d --name wisecow -p 8080:4499 britneycorreia/wisecow-app:latest

4. Pushing Docker Image to Docker Hub

After successfully building the Docker image, it was pushed to Docker Hub:

docker tag britneycorreia/wisecow-app:latest britneycorreia/wisecow-app:latest

docker push britneycorreia/wisecow-app:latest

This allowed the image to be accessed externally for use in the Kubernetes deployment.

5. Kubernetes Setup

The application was deployed on a Kubernetes cluster with the following configurations:

a. Deployment

A deployment YAML file (deployment.yaml) was created to define the pods and specify the number of replicas:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wisecow-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: wisecow
  template:
    metadata:
      labels:
        app: wisecow
    spec:
      containers:
      - name: wisecow
        image: britneycorreia/wisecow-app:latest
```

```
    ports:
    - containerPort: 4499
```

This ensured that three replicas of the application were running, providing high availability.

b. Scaling

The deployment was scaled as follows:

```
kubectl scale deployment wisecow-deployment --replicas=3
```

c. Service

A LoadBalancer service was created to expose the application on port 8080:

```
apiVersion: v1
kind: Service
metadata:
  name: wisecow-service
spec:
  type: LoadBalancer
  ports:
  - port: 8080
    targetPort: 4499
  selector:
    app: wisecow
```

This service allowed access to the application externally on the designated port.

6. Continuous Integration (CI) Pipeline

The CI pipeline was implemented in GitHub Actions with a .github/workflows/ci.yml file:

- **Trigger**: The pipeline is triggered on pushes and pull requests to the main branch.
- **Build and Test**: It pulls the latest code, builds the Docker image, and tests it to ensure functionality.

CI Configuration

```
name: CI Pipeline
on:
  push:
    branches:
```

```
    - main
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v2

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1

      - name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Build and Push Docker Image
        run: |
          docker build -t britneycorreia/wisecow-app:latest .
          docker push britneycorreia/wisecow-app:latest
```

7. Continuous Deployment (CD) Pipeline

The CD pipeline (.github/workflows/cd.yml) was configured to automate deployment after CI:

- **Trigger**: Activated when a new Docker image is pushed.
- **Deploy to Kubernetes**: Uses kubectl to apply the latest image to the Kubernetes deployment.

CD Configuration

```
name: CD Pipeline

on:
  push:
    tags:
      - "v*.*.*"


jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Code
        uses: actions/checkout@v2


      - name: Set up Kubernetes
        uses: azure/setup-kubectl@v1
        with:
          version: v1.18.0


      - name: Deploy to Kubernetes
        env:
          KUBECONFIG: ${{ secrets.KUBECONFIG }}
        run: |
          kubectl apply -f deployment.yaml
          kubectl apply -f service.yaml
```

8. Testing and Verification

- **Access Test**: Verified access via localhost:8080 to ensure the application was running as expected.
- **Pods and Service Check**: Used kubectl get pods and kubectl get svc to confirm pod status and service connectivity.
- **Scaling Test**: Successfully scaled the deployment to handle more instances.

9. Challenges and Solutions

- **Issue**: Localhost connection issues on port 8080.
    - **Solution**: Adjusted Kubernetes service configuration to match the correct ports.
- **Issue**: Errors with GitHub Actions permissions when pushing to a shared repository.
    - **Solution**: Created a personal repository for project files to test and validate CI/CD independently.

10. Conclusion

This project successfully demonstrated the deployment of a containerized application with automated CI/CD on a Kubernetes cluster. Key learnings include handling Docker images, creating Kubernetes deployments and services, and setting up a CI/CD pipeline. This end-to-end deployment pipeline ensures the application is ready for production with minimal manual intervention.