Accuknox DevOps Trainee Practical Assessment

Problem Statement 1:

Project Repository: - https://github.com/nyrahul/wisecow Wisecow App

Requirements

1. **Dockerization**
    - Develop a Dockerfile for creating a container image of the Wisecow application.
2. **Kubernetes Deployment**
    - Craft Kubernetes manifest files for deploying the Wisecow application in a Kubernetes environment.
    - The Wisecow app must be exposed as a Kubernetes service for accessibility.
3. **Continuous Integration and Deployment (CI/CD)**
    - Implement a GitHub Actions workflow for:
        - Automating the build and push of the Docker image to a container registry whenever changes are committed to the repository.
        - Continuous Deployment: Automatically deploy the updated application to the Kubernetes environment following successful image builds.
4. **TLS Implementation**
    - Ensure that the Wisecow application supports secure TLS communication.

Expected Artifacts

- A private GitHub repository containing:
    - The Wisecow application source code.
    - The Dockerfile for the application.
    - Kubernetes manifest files for deployment.
    - The CI/CD pipeline configuration.
    - A GitHub Actions workflow file for facilitating Continuous Build and Deployment (CI/CD).

Access Control

- The GitHub repository should be set to public.

End Goal

The successful containerization and deployment of the Wisecow application to the Kubernetes environment with an automated CI/CD pipeline and secured with TLS communication.

Step-by-Step Guide

Step 1: Dockerization

1. Clone the Repository

git clone https://github.com/nyrahul/wisecow.git

cd wisecow

```
controlplane $ git clone https://github.com/nyrahul/wisecow.git
Cloning into 'wisecow'...
remote: Enumerating objects: 28, done.
remote: Total 28 (delta 0), reused 0 (delta 0), pack-reused 28
Unpacking objects: 100% (28/28), 10.51 KiB | 1.31 MiB/s, done.
controlplane $ cd wisecow
controlplane $ ls
LICENSE  README.md  wisecow.sh
```

2. Create a Dockerfile

In the root directory of the Wisecow application, create a file named Dockerfile with the following content:

vi Dockerfile

```
FROM ubuntu:20.04

RUN apt-get update && \

    apt-get install -y \

    netcat \

    fortune-mod \

    cowsay \

    && rm -rf /var/lib/apt/lists/*

WORKDIR /app

COPY wisecow.sh /app/wisecow.sh

RUN chmod +x /app/wisecow.sh

EXPOSE 4499

ENV PATH="/usr/games:${PATH}"

CMD ["/app/wisecow.sh"]
```

3. Build the Docker Image

docker build -t pratikmule127/wisecow:latest .

```
Successfully built 42e5484c270c
Successfully tagged pratikmule127/wisecow:latest
```

4. Push the Image to Docker Hub

docker login

docker tag wisecow-app pratikmule127/wisecow-app:latest

docker push pratikmule127/wisecow-app:latest

docker run -d -p 4499:4499 pratikmule127/wisecow-app:latest

```
controlplane $ docker run -d -p 4499:4499 pratikmule127/wisecow-app:latest
Unable to find image 'pratikmule127/wisecow-app:latest' locally
latest: Pulling from pratikmule127/wisecow-app
9ea8908f4765: Already exists
94af23154fd3: Pull complete
d830597639ff: Pull complete
11ee3884b6b3: Pull complete
Digest: sha256:ad908281cf360c79f05bf983cc12fffd99779e0386d5ebb7c0c1a6da3c19a869
Status: Downloaded newer image for pratikmule127/wisecow-app:latest
2daa3d704191477c5efa457755a0c165473361af7afa2fc2250c9c0019790ff7
controlplane $
```

```
controlplane $ docker ps -a
CONTAINER ID   IMAGE                          COMMAND            CREATED          STATUS          PORTS                                       NAMES
2daa3d704191   pratikmule127/wisecow-app:latest   "/app/wisecow.sh"   21 seconds ago   Up 20 seconds   0.0.0.0:4499->4499/tcp, :::4499->4499/tcp   hardcore_burnell
```

d8f46986-4a89-4aa7-9d3f-b212aeea06d0-10-244-7-197-4499.papa.r.killercoda.com

```
 _____________________________________
/ Try the Moo Shu Pork. It is especially \
\ good today.                            /
 -------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Step 2: Kubernetes Deployment

1. Create Kubernetes Manifests

1. create the following files:

- vi deployment.yaml

```
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: pratikmule127/wisecow:latest
        ports:
        - containerPort: 80
```

- vi service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: wisecow-service
spec:
  type: NodePort
  selector:
    app: wisecow
  ports:
    - protocol: TCP
      port: 80
      targetPort: 4499
      nodePort: 32092
```

2. Apply the Manifests to Kubernetes

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

kubectl get deployments

kubectl get services

```
controlplane $ kubectl apply -f deployment.yaml
deployment.apps/wisecow-deployment unchanged
controlplane $ kubectl apply -f service.yaml
service/wisecow-service unchanged
controlplane $ kubectl get deployments
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
wisecow-deployment   3/3     3            3           28s
controlplane $ kubectl get services
NAME             TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
kubernetes       ClusterIP   10.96.0.1     <none>        443/TCP        24d
wisecow-service  NodePort    10.101.82.6   <none>        80:32092/TCP   6m30s
controlplane $
```

Step 3: CI/CD Pipeline with GitHub Actions

1. Create GitHub Actions Workflow

In your GitHub repository, create a directory .github/workflows and a file named ci-cd.yaml with the following content:

- vi ci-cd.yaml

```
name: CI/CD Pipeline

on:

  push:

    branches:

      - main

jobs:

  build:

    runs-on: ubuntu-latest

    steps:

      - name: Checkout code

        uses: actions/checkout@v3

      - name: Set up Docker Buildx

        uses: docker/setup-buildx-action@v2

      - name: Cache Docker layers

        uses: docker/cache-action@v2

        with:

          path: /tmp/.buildx-cache

      - name: Build and push Docker image

        uses: docker/build-push-action@v3

        with:

          context: .

          file: Dockerfile

          push: true
```

```
      tags: pratikmule127/wisecow:latest
    - name: Set up kubectl
      uses: azure/setup-kubectl@v1
      with:
        version: '1.25.0'
    - name: Deploy to Kubernetes
      run: |
        kubectl config set-cluster my-cluster --server=https://172.30.1.2:6443 --certificate-authority=/path/to/ca.crt
        kubectl config set-credentials my-user --token=${{ secrets.KUBE_TOKEN }}
        kubectl config set-context my-context --cluster=my-cluster --user=my-user --namespace=default
        kubectl config use-context my-context
        kubectl apply -f deployment.yaml
        kubectl apply -f service.yaml
```

2. Set Up Secrets in GitHub Repository

- **Go to the Settings of your repository.**
- **Under "Secrets and variables," add the following secrets:**
  - **DOCKER_HUB_USERNAME: Your Docker Hub username.**
  - **DOCKER_HUB_ACCESS_TOKEN: Your Docker Hub access token.**
  - **KUBE_TOKEN: Your Kubernetes access token.**

Step 4: TLS Implementation

1. Install Cert-Manager

kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.5.3/cert-manager.yaml

2. Create Issuer

vi issuer.yaml

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: letsencrypt-staging
spec:
  acme:
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    email: your-email@example.com
    privateKeySecretRef:
      name: wisecow-issuer
    solvers:
    - http01:
        ingress:
          class: nginx
```

kubectl apply -f issuer.yaml

```
controlplane $ kubectl apply -f issuer.yaml
issuer.cert-manager.io/letsencrypt-staging created
```

3. **Create Ingress Resource**

vi ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: wisecow-ingress
  annotations:
    cert-manager.io/issuer: "letsencrypt-staging"
spec:
  rules:
  - host: wisecow.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: wisecow-service
            port:
              number: 80
  tls:
  - hosts:
    - wisecow.example.com
    secretName: wisecow-tls
```

kubectl apply -f ingress.yaml

```
controlplane $ kubectl apply -f ingress.yaml
ingress.networking.k8s.io/wisecow-ingress created
```

Links and Credentials

- **GitHub Repository**: https://github.com/pratikmule127/ci-cd-pipeline.git
- **Docker Hub Image**: pratikmule127/wisecow

Problem Statement 2:

1. System Health Monitoring Script: Develop a script that monitors the health of a Linux system. It should check CPUusage, memory usage, disk space, and running processes. If any of these metrics exceed predefined thresholds (e.g., CPU usage > 80%), the script should send an alert to the console or a log file.

vi system_health.sh

```
[root@localhost ~]# vi system_health.sh
```

```
#!/bin/bash


CPU_THRESHOLD=80
MEMORY_THRESHOLD=80
DISK_THRESHOLD=80
CPU_USAGE=$(top -bn1 | grep "Cpu(s)" | awk '{print $2 + $4}')
MEMORY_USAGE=$(free | grep Mem | awk '{print $3/$2 * 100.0}')
DISK_USAGE=$(df / | grep / | awk '{print $5}' | sed 's/%//g')
RUNNING_PROCESSES=$(ps -e | wc -l)
log_alert() {
   echo "$(date): $1" >> /var/log/system_health.log
}
if (( $(echo "$CPU_USAGE > $CPU_THRESHOLD" | bc -l) )); then
   log_alert "High CPU usage: $CPU_USAGE%"
fi
if (( $(echo "$MEMORY_USAGE > $MEMORY_THRESHOLD" | bc -l) )); then
   log_alert "High Memory usage: $MEMORY_USAGE%"
fi
if [ $DISK_USAGE -gt $DISK_THRESHOLD ]; then
   log_alert "High Disk usage: $DISK_USAGE%"
fi
log_alert "Running processes: $RUNNING_PROCESSES"
echo "System health check complete. Check /var/log/system_health.log for details."
```

To run these script, follow these steps:

Create the Script Files: Save each script into a separate file with a .sh extension. For example:

- System Health Monitoring Script: system_health.sh

```
[root@localhost ~]# vi system_health.sh
```

Make the Scripts Executable: Use the chmod command to make each script executable. For example:

chmod +x system_health.sh

```
[root@localhost ~]# chmod +x system_health.sh
```

Run the Scripts: Execute the scripts using the ./ command followed by the script name. For example:

./system_health.sh

```
[root@localhost ~]# ./system_health.sh
System health check complete. Check /var/log/system_health.log for details.
[root@localhost ~]#
```

View the Log File: You can use commands like cat to view the contents of the log file.

cat /var/log/system_health.log

```
[root@localhost ~]# cat /var/log/system_health.log
Sun Jul 28 06:45:04 PM IST 2024: Running processes: 323
Sun Jul 28 07:04:09 PM IST 2024: Running processes: 353
```

4. Application Health Checker: Please write a script that can check the uptime of an application and determine if it is functioning correctly or not. The script must accurately assess the application's status by checking HTTP status codes. It should be able to detect if the application is 'up', meaning it is functioning correctly, or 'down', indicating that it is unavailable or not responding.

vi app_health_checker.sh

```
root@rhel:~# vi app_health_checker.sh
```

```
#!/bin/bash


APP_URL="http://your-application-url"

LOG_FILE="/var/log/app_health.log"

HTTP_STATUS=$(curl -o /dev/null -s -w "%{http_code}\n" $APP_URL)

log_status() {

   echo "$(date): $1" >> $LOG_FILE

}

if [ $HTTP_STATUS -eq 200 ]; then

   log_status "Application is UP"

else

   log_status "Application is DOWN (Status Code: $HTTP_STATUS)"

fi

echo "Application health check complete. Check $LOG_FILE for details."
```

To run these script, follow these steps:

Create the Script Files: Save each script into a separate file with a .sh extension. For example:

- Application Health Checker: app_health_checker.sh

```
root@rhel:~# vi app_health_checker.sh
```

Make the Scripts Executable: Use the chmod command to make each script executable. For example:

chmod +x app_health_checker.sh

```
root@rhel:~# chmod +x app_health_checker.sh
```

Run the Scripts: Execute the scripts using the ./ command followed by the script name. For example:

./app_health_checker.sh

```
root@rhel:~# ./app_health_checker.sh
Application health check complete. Check /var/log/app_health.log for details.
root@rhel:~#
```