**Problem Statement 1**

**1. Dockerization**

We need to create a Docker file to containerize the **Wise Cow** application.

Step 1: Use a lightweight base image

FROM python:3.9-slim

Step 2: Set working directory in the container

WORKDIR /app

Step 3: Copy the application source code into the container

COPY. /app

Step 4: Install dependencies (assuming requirements.txt is present)

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

Step 5: Expose the port on which the application runs (adjust based on the app)

EXPOSE 8000

Step 6: Run the application

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

### ****2. Kubernetes Deployment****

We need to create the following Kubernetes YAML manifest files:

* Deployment YAML for deploying the Wise cow app.
* Service YAML for exposing the app.
* Optionally, an Ingress YAML for external access with TLS.

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-container

image: <your-dockerhub-username>/wisecow:latest

ports:

- containerPort: 8000

Kubernetes Service YAML

apiVersion: v1

kind: Service

metadata:

name: wisecow-service

spec:

selector:

app: wisecow

ports:

- protocol: TCP

port: 80

targetPort: 8000

type: LoadBalancer

**Kubernetes Ingress YAML (with TLS)**:

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: wisecow-ingress

annotations:

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

spec:

tls:

- hosts:

- wisecow.yourdomain.com

secretName: wisecow-tls-secret

rules:

- host: wisecow.yourdomain.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: wisecow-service

port:

number: 80

3. CI/CD Pipeline with GitHub Actions

name: CI/CD Pipeline

on:

push:

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 DockerHub

run: echo "${{ secrets.DOCKER\_PASSWORD }}" | docker login -u "${{ secrets.DOCKER\_USERNAME }}" --password-stdin

- name: Build and Push Docker image

run: |

docker build -t ${{ secrets.DOCKER\_USERNAME }}/wisecow:latest .

docker push ${{ secrets.DOCKER\_USERNAME }}/wisecow:latest

deploy:

runs-on: ubuntu-latest

needs: build

steps:

- name: Set up kubectl

uses: azure/setup-kubectl@v1

with:

version: 'latest'

- name: Deploy to Kubernetes

run: |

kubectl apply -f k8s/deployment.yaml

kubectl apply -f k8s/service.yaml

kubectl apply -f k8s/ingress.yaml

4. TLS Implementation

**1. Install cert-manager** in your Kubernetes cluster:

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

**2. Create a ClusterIssuer for Let’s Encrypt**:

apiVersion: cert-manager.io/v1

kind: ClusterIssuer

metadata:

name: letsencrypt-prod

spec:

acme:

server: https://acme-v02.api.letsencrypt.org/directory

email: your-email@example.com

privateKeySecretRef:

name: letsencrypt-prod-key

solvers:

- http01:

ingress:

class: nginx

**3. Update Ingress to use cert-manager**: Update the Ingress file to obtain and use a TLS certificate automatically:

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: wisecow-ingress

annotations:

cert-manager.io/cluster-issuer: letsencrypt-prod

spec:

tls:

- hosts:

- wisecow.yourdomain.com

secretName: wisecow-tls-secret

rules:

- host: wisecow.yourdomain.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: wisecow-service

port:

number: 80

**Problem Statement 2**

### ****1. Log File Analyser (Objective 3)****

This Bash script analyses a web server log (e.g., Apache, Nginx) and provides a summary report of:

* Number of 404 errors.
* Most requested pages.
* IP addresses with the most requests.

#### **Bash Script:** log\_analyzer.sh

#!/bin/bash

# Log file location (you can modify this to match the actual log file path)

LOG\_FILE="/var/log/nginx/access.log"

# Check if the log file exists

if [[ ! -f $LOG\_FILE ]]; then

echo "Log file not found: $LOG\_FILE"

exit 1

fi

# Number of 404 errors

echo "Number of 404 errors:"

grep " 404 " $LOG\_FILE | wc -l

echo ""

# Top 10 most requested pages

echo "Top 10 most requested pages:"

awk '{print $7}' $LOG\_FILE | sort | uniq -c | sort -nr | head -10

echo ""

# Top 10 IP addresses with the most requests

echo "Top 10 IP addresses with the most requests:"

awk '{print $1}' $LOG\_FILE | sort | uniq -c | sort -nr | head -10

To run the script:

chmod +x log\_analyzer.sh

./log\_analyzer.sh

### ****2. Application Health Checker (Objective 4)****

This Bash script checks the uptime and HTTP status code of an application by periodically sending HTTP requests to the server and determining if the application is "up" (200 OK) or "down."

#### **Bash Script:** health\_checker.sh

#!/bin/bash

# Application URL (replace with the actual application URL)

APP\_URL="http://localhost:8000"

# HTTP status code check

status\_code=$(curl --write-out "%{http\_code}" --silent --output /dev/null "$APP\_URL")

# Uptime check (Check if the server is up or down)

if [[ $status\_code -eq 200 ]]; then

echo "Application is UP and running. Status Code: 200 OK"

else

echo "Application is DOWN. Status Code: $status\_code"

fi

To run the script:

chmod +x health\_checker.sh

./health\_checker.sh

**Log File Analyser Output**:

Number of 404 errors:

23

Top 10 most requested pages:

35 /index.html

30 /about

28 /contact

15 /home

10 /products

9 /blog

7 /services

5 /pricing

4 /login

2 /faq

Top 10 IP addresses with the most requests:

45 192.168.1.2

30 192.168.1.3

28 192.168.1.4

20 192.168.1.5

15 192.168.1.6

10 192.168.1.7

7 192.168.1.8

5 192.168.1.9

3 192.168.1.10

2 192.168.1.11

**Health Checker Output**:

Application is UP and running. Status Code: 200 OK

Or

Application is DOWN. Status Code: 500