Problem Statement 1:

Title: Containerisation and Deployment of Wisecow Application on Kubernetes

To containerize and deploy the Wisecow application on Kubernetes, you can follow these general steps:

### 1. ****Dockerization****

First, create a Dockerfile to containerize the Wisecow application.

# Use an official Python runtime as a parent image

FROM python: 3.9-slim

# set the working directory in the container

WORKDIR /app

# Copy the current directory contents into the container at /app

COPY . /app

# Install any needed packages specified in requirements.txt

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

# Make port 80 available to the world outside this container

EXPOSE 80

# Define environment variable

ENV NAME Wisecow

# Run app.py when the container launches

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

Build the Docker image:

docker build -t wisecow-app .

Test the Docker image locally:

docker run -p 8080:80 wisecow-app

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

Create Kubernetes manifest files for deployment.

**Deployment YAML (wisecow-deployment.yaml):**

apiVersion: apps/v1

kind: Deployment

metadata:

name: wisecow-deployment

labels:

app: wisecow

spec:

replicas: 1

selector:

matchLabels:

app: wisecow

template:

metadata:

labels:

app: wisecow

spec:

containers:

- name: wisecow

image: <your-container-registry>/wisecow-app:latest

ports:

- containerPort: 80

**Service YAML (wisecow-service.yaml):**

apiVersion: v1

kind: Service

metadata:

name: wisecow-service

spec:

selector:

app: wisecow

ports:

- protocol: TCP

port: 80

targetPort: 80

type: LoadBalancer

Apply the manifest files:

kubectl apply -f wisecow-deployment.yaml

kubectl apply -f wisecow-service.yaml

### 3. ****Continuous Integration and Deployment (CI/CD)****

Set up a CI/CD pipeline using GitHub Actions.

**GitHub Actions Workflow (.github/workflows/ci-cd.yml):**

name: CI/CD Pipeline

on:

push:

branches:

- main

jobs:

build-and-deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Login to Container Registry

uses: docker/login-action@v1

with:

username: ${{ secrets.REGISTRY\_USERNAME }}

password: ${{ secrets.REGISTRY\_PASSWORD }}

- name: Build Docker image

run: docker build -t <your-container-registry>/wisecow-app:latest .

- name: Push Docker image

run: docker push <your-container-registry>/wisecow-app:latest

- name: Deploy to Kubernetes

run: kubectl apply -f wisecow-deployment.yaml

env:

KUBECONFIG: ${{ secrets.KUBECONFIG }}

Problem Statement 2:

1. System Health Monitoring Script:

First, you’ll need to install the psutil library if you haven’t already:

pip install psutil

The script as follows.

import psutil

import datetime

def log\_system\_health():

timestamp = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

cpu\_usage = psutil.cpu\_percent(interval=1)

memory\_usage = psutil.virtual\_memory().percent

disk\_usage = psutil.disk\_usage('/').percent

log\_entry = (

f"{timestamp} - CPU Usage: {cpu\_usage}%\n"

f"{timestamp} - Memory Usage: {memory\_usage}%\n"

f"{timestamp} - Disk Usage: {disk\_usage}%\n"

)

with open("system\_health\_log.txt", "a") as log\_file:

log\_file.write(log\_entry)

if \_\_name\_\_ == "\_\_main\_\_":

log\_system\_health()

This script will create a log file named system\_health\_log.txt in the same directory as the script, recording the CPU, memory, and disk usage with timestamps each time it runs12

Sure! Here’s a simple Bash script that monitors the health of a Linux system by checking CPU usage, memory usage, disk space, and running processes. If any of these metrics exceed predefined thresholds, the script will log an alert to the console.

#!/bin/bash

# Thresholds

CPU\_THRESHOLD=80

MEM\_THRESHOLD=80

DISK\_THRESHOLD=80

# Function to check CPU usage

check\_cpu() {

CPU\_USAGE=$(top -bn1 | grep "Cpu(s)" | sed "s/.\*, \*\([0-9.]\*\)%\* id.\*/\1/" | awk '{print 100 - $1}')

if (( $(echo "$CPU\_USAGE > $CPU\_THRESHOLD" | bc -l) )); then

echo "ALERT: CPU usage is above $CPU\_THRESHOLD%: Current usage is $CPU\_USAGE%" | tee -a /var/log/system\_health.log

fi

}

# Function to check memory usage

check\_memory() {

MEM\_USAGE=$(free | grep Mem | awk '{print $3/$2 \* 100.0}')

if (( $(echo "$MEM\_USAGE > $MEM\_THRESHOLD" | bc -l) )); then

echo "ALERT: Memory usage is above $MEM\_THRESHOLD%: Current usage is $MEM\_USAGE%" | tee -a /var/log/system\_health.log

fi

}

# Function to check disk space usage

check\_disk() {

DISK\_USAGE=$(df / | grep / | awk '{ print $5}' | sed 's/%//g')

if [ $DISK\_USAGE -gt $DISK\_THRESHOLD ]; then

echo "ALERT: Disk space usage is above $DISK\_THRESHOLD%: Current usage is $DISK\_USAGE%" | tee -a /var/log/system\_health.log

fi

}

# Function to check running processes

check\_processes() {

RUNNING\_PROCESSES=$(ps aux --no-heading | wc -l)

echo "INFO: Number of running processes: $RUNNING\_PROCESSES" | tee -a /var/log/system\_health.log

}

# Main function to run all checks

main() {

echo "System Health Check - $(date)" | tee -a /var/log/system\_health.log

check\_cpu

check\_memory

check\_disk

check\_processes

echo "----------------------------------------" | tee -a /var/log/system\_health.log

}

# Run the main function

main

This script performs the following checks:

CPU usage: If the CPU usage exceeds 80%, it logs an alert.

Memory usage: If the memory usage exceeds 80%, it logs an alert.

Disk space usage: If the disk space usage exceeds 80%, it logs an alert.

Running processes: Logs the number of running processes.

You can adjust the thresholds as needed by changing the values of CPU\_THRESHOLD, MEM\_THRESHOLD, and DISK\_THRESHOLD.

To run this script, save it to a file (e.g., system\_health.sh), make it executable (chmod +x system\_health.sh), and then execute it (./system\_health.sh). The alerts will be logged to the console and to /var/log/system\_health.log.

2. Automated Backup Solution:

import os

import subprocess

import datetime

# Configuration

local\_directory = "/path/to/local/directory"

remote\_directory = "/path/to/remote/directory"

remote\_server = "user@remote\_server\_address"

log\_file = "/path/to/log\_file.log"

def backup\_directory():

try:

# Create a timestamp for the log

timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

# Run the rsync command

result = subprocess.run(

["rsync", "-avz", local\_directory, f"{remote\_server}:{remote\_directory}"],

capture\_output=True,

text=True

)

# Check if the rsync command was successful

if result.returncode == 0:

status = "SUCCESS"

message = f"Backup completed successfully at {timestamp}\n"

else:

status = "FAILURE"

message = f"Backup failed at {timestamp}\nError: {result.stderr}\n"

# Log the result

with open(log\_file, "a") as log:

log.write(f"{timestamp} - {status}\n")

log.write(message)

print(message)

except Exception as e:

# Log any exceptions that occur

with open(log\_file, "a") as log:

log.write(f"{timestamp} - EXCEPTION\n")

log.write(f"Exception occurred: {str(e)}\n")

print(f"Exception occurred: {str(e)}")

if \_\_name\_\_ == "\_\_main\_\_":

backup\_directory()