Accuknox QA Engineer Practical Assessment

## **Problem Statement 1**:

##### Title: Containerisation and Deployment of Wisecow Application on Kubernetes

**Objective:** To containerize and deploy the Wisecow application, hosted in the above-mentioned GitHub repository, on a Kubernetes environment with secure TLS communication.

**Answer:**

##### Dockerization

1. Develop a Dockerfile for creating a container image of the Wisecow application:

|  |
| --- |
| FROM python:3.9-slim  WORKDIR /app  COPY . /app  RUN pip install --no-cache-dir -r requirements.txt  EXPOSE 80  ENV NAME Wisecow  CMD ["python", "app.py"] |

##### Kubernetes Deployment

1. Create Kubernetes manifest files for deploying the Wisecow application in a Kubernetes environment:

Directory Structure:

|  |
| --- |
| mkdir k8s  cd k8s |

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: <your-dockerhub-username>/wisecow:latest  ports:  - containerPort: 80 |

service.yaml:

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: wisecow-service  spec:  selector:  app: wisecow  ports:  - protocol: TCP  port: 80  targetPort: 80  type: LoadBalancer |

##### Continuous Integration and Deployment (CI/CD)

1. Implement a GitHub Actions workflow for automating the build and push of the Docker image to a container registry and continuous deployment:

Directory Structure:

|  |
| --- |
| mkdir -p .github/workflows |

ci-cd.yaml:

|  |
| --- |
| 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: Log in to Docker Hub  uses: docker/login-action@v1  with:  username: ${{ secrets.DOCKER\_USERNAME }}  password: ${{ secrets.DOCKER\_PASSWORD }}  - name: Build and push Docker image  run: |  docker build . -t ${{ secrets.DOCKER\_USERNAME }}/wisecow:latest  docker push ${{ secrets.DOCKER\_USERNAME }}/wisecow:latest  deploy:  needs: build  runs-on: ubuntu-latest  environment:  name: production  url: http://<your-k8s-cluster-ip>  steps:  - name: Set up K8s context  uses: azure/k8s-set-context@v1  with:  method: kubeconfig  kubeconfig: ${{ secrets.KUBECONFIG }}  - name: Deploy to Kubernetes  run: |  kubectl apply -f k8s/deployment.yaml  kubectl apply -f k8s/service.yaml |

##### TLS Implementation

1. Ensure that the Wisecow application supports secure TLS communication using Let's Encrypt and Kubernetes Ingress:

Install cert-manager:

|  |
| --- |
| kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.8.0/cert-manager.yaml |

Create an Issuer for Let's Encrypt:

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

Create an Ingress resource:

|  |
| --- |
| apiVersion: networking.k8s.io/v1  kind: Ingress  metadata:  name: wisecow-ingress  annotations:  cert-manager.io/issuer: "letsencrypt-prod"  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 |

**Note :** I have completed the above implementation using an AI tool. I only have theoretical knowledge about Docker and Kubernetes; I do not have working experience with them.

## **Problem Statement 2**:

##### Log File Analyzer:

Create a script that analyzes web server logs (e.g., Apache, Nginx) for common patterns such as the number of 404 errors, the most requested pages, or IP addresses with the most requests. The script should output a summarized report.

|  |
| --- |
| import re  from collections import defaultdict, Counter  # Function to parse the log file  def parse\_log\_line(line):  parts = line.split()  ip = parts[0]  request = parts[5] + ' ' + parts[6] + ' ' + parts[7]  status\_code = parts[8]  return ip, request, status\_code  def analyze\_log\_file(log\_file\_path):  ip\_counter = Counter()  request\_counter = Counter()  error\_404\_counter = 0  with open(log\_file\_path, 'r') as file:  for line in file:  ip, request, status\_code = parse\_log\_line(line)  ip\_counter[ip] += 1  request\_counter[request] += 1  if status\_code == '404':  error\_404\_counter += 1  most\_common\_ips = ip\_counter.most\_common(10)  most\_requested\_pages = request\_counter.most\_common(10)  return error\_404\_counter, most\_requested\_pages, most\_common\_ips  def print\_report(error\_404\_count, most\_requested\_pages, most\_common\_ips):  print("Summary Report:")  print(f"Number of 404 errors: {error\_404\_count}\n")  print("Most Requested Pages:")  for page, count in most\_requested\_pages:  print(f"{page}: {count} requests")  print("\nIP Addresses with Most Requests:")  for ip, count in most\_common\_ips:  print(f"{ip}: {count} requests")  if \_\_name\_\_ == "\_\_main\_\_":  log\_file\_path = 'fake/path /logfile.log' # Replace path with original log file path  error\_404\_count, most\_requested\_pages, most\_common\_ips = analyze\_log\_file(log\_file\_path)  print\_report(error\_404\_count, most\_requested\_pages, most\_common\_ips) |

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.

|  |
| --- |
| import requests  import time  def check\_application\_health(url, timeout=5):  try:  response = requests.get(url, timeout=timeout)  if response.status\_code == 200:  return "up"  else:  return "down"  except requests.exceptions.RequestException as e:  return "down"  def main():  url = 'http://health-application-url.com' # Replace with health application's URL  check\_interval = 60  while True:  status = check\_application\_health(url)  print(f"Application status: {status}")  time.sleep(check\_interval)  if \_\_name\_\_ == "\_\_main\_\_":  main() |