Accuknox QA Trainee Practical Assessment - Detailed Solutions

Problem Statement 1: Containerization and Deployment of Wisecow Application on Kubernetes

Step-by-Step Solution

1. Dockerization

   - Create a `Dockerfile` for the Wisecow application to containerize it.

   Dockerfile:

   ```dockerfile
   FROM python:3.9-slim
   WORKDIR /app
   COPY . .
   RUN pip install --no-cache-dir -r requirements.txt
   CMD ["python", "app.py"]
   ```

   - This `Dockerfile` uses Python 3.9, sets the working directory to `/app`, copies the source code into the container, installs dependencies, and runs the application with the `python app.py` command.

2. Kubernetes Deployment YAML Files

   - Create the necessary Kubernetes manifests to deploy the application.

   deployment.yaml:

   ```yaml
   apiVersion: apps/v1
   ```

```yaml
kind: Deployment
metadata:
  name: wisecow-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: wisecow
  template:
    metadata:
      labels:
        app: wisecow
    spec:
      containers:
      - name: wisecow-container
        image: your-docker-repo/wisecow:latest
        ports:
        - containerPort: 80
```

service.yaml:

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

- The `deployment.yaml` specifies the application deployment, including replicas, labels, and the container image.
- The `service.yaml` creates a service to expose the application and allows access through a LoadBalancer.

3. CI/CD Pipeline with GitHub Actions

- Create a GitHub Actions workflow file to automate building and deploying the Docker image.

.github/workflows/ci-cd.yaml:

```yaml
name: CI/CD Pipeline

on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout Code
      uses: actions/checkout@v3

    - name: Set up Docker
      uses: docker/setup-buildx-action@v2

    - 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 your-docker-repo/wisecow:latest .
        docker push your-docker-repo/wisecow:latest

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

- This workflow triggers on every push to the `main` branch, builds the Docker image, pushes it to a registry, and deploys it to the Kubernetes cluster.

4. TLS Implementation

   - Use **Cert-Manager** or configure TLS certificates manually to secure the application. Integrate these certificates into the Kubernetes configuration for HTTPS support.

Problem Statement 2: Bash/Python Scripting Tasks

Option 1: System Health Monitoring Script (Python)

```python
import psutil

def check_system_health():
    cpu_usage = psutil.cpu_percent(interval=1)
    memory_info = psutil.virtual_memory()
    disk_usage = psutil.disk_usage('/')

    if cpu_usage > 80:
        print(f"Alert! High CPU usage: {cpu_usage}%")
    if memory_info.percent > 80:
        print(f"Alert! High Memory usage: {memory_info.percent}%")
    if disk_usage.percent > 80:
        print(f"Alert! Low Disk Space: {disk_usage.percent}% used")

    print(f"CPU Usage: {cpu_usage}%, Memory Usage: {memory_info.percent}%, Disk Usage: {disk_usage.percent}%")
```

```
if __name__ == "__main__":
    check_system_health()
```

- This script uses the `psutil` library to check system CPU, memory, and disk usage, and prints alerts if thresholds are exceeded.

Option 2: Log File Analyzer (Python)

```python
import re
from collections import Counter


def analyze_log(file_path):
    with open(file_path, 'r') as log_file:
        logs = log_file.readlines()


    error_404_count = len([line for line in logs if "404" in line])
    ip_addresses = [re.search(r'(\d+\.\d+\.\d+\.\d+)', line).group() for line in logs if re.search(r'(\d+\.\d+\.\d+\.\d+)', line)]


    most_common_ips = Counter(ip_addresses).most_common(5)
    print(f"404 Errors: {error_404_count}")
    print("Top 5 IP addresses:")
    for ip, count in most_common_ips:
        print(f"{ip}: {count} requests")


if __name__ == "__main__":
    analyze_log('server.log')
```

- This script analyzes server logs for 404 errors and counts IP addresses, displaying the top 5 most common IPs.