Problem Solution 1

To achieve the objective of containerizing and deploying the Wisecow application on a Kubernetes environment with secure TLS communication, we will follow a structured approach. Below are the steps and artifacts required to complete the task

STEP 1: Dockerization

1. **Create a Dockerfile:** This file will define how to build the Docker image for the Wisecow application.Below is a sample Dockerfile assuming the application is a Node.js app.Adjust the base image and commands according to your application's requirements.

```
# Use the official Node.js image as a base
FROM node:14

# Set the working directory
WORKDIR /usr/src/app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application code
COPY . .

# Expose the application port
EXPOSE 3000

# Command to run the application
CMD ["npm", "start"]
```

STEP 2: Kubernetes Deployment

2. Create Kubernetes Manifest Files: You will need a deployment and a service manifest file. Below are examples of both.

Deployment Manifest('wisecow-deployment.yaml'):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wisecow
spec:
  replicas: 3
  selector:
    matchLabels:
      app: wisecow
  template:
    metadata:
      labels:
        app: wisecow
    spec:
      containers:
      - name: wisecow
        image: <your-docker-registry>/wisecow:latest
        ports:
        - containerPort: 3000
```

Service Manifest(‘wisecow-service.yaml’):

```
apiVersion: v1
kind: Service
metadata:
  name: wisecow
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 3000
  selector:
    app: wisecow
```

STEP 3: Continuous Integration and Deployment(CI/CD)

3. Implement GitHub Actions Workflows: Create a ‘.github/workflows/ci-cd.yml’ file to automate the build and deployment process.

```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
      uses: docker/build-push-action@v2
      with:
        context: .
        push: true
        tags: <your-docker-registry>/wisecow:latest

  deploy:
    runs-on: ubuntu-latest
    needs: build
    steps:
    - name: Set up kubectl
      uses: azure/setup-kubectl@v1
      with:
        version: 'latest'

    - name: Configure kubectl
```

```
        run: |
          echo "${{ secrets.KUBE_CONFIG }}" > kubeconfig
          export KUBECONFIG=kubeconfig

      - name: Deploy to Kubernetes
        run: |
          kubectl apply -f wisecow-deployment.yaml
          kubectl apply -f wisecow-service.yaml
```

STEP 4: TLS I mplementation

4. Ensure TLS Communication: You can use a tool like Cert-Manager to manage TLS certificate in kubernetes. Below is a basic example of how to set it up.

Install Cert- Manager

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

Create a Certificate Resource:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: wisecow-cert
spec:
  secretName: wisecow-tls
  issuerRef:
    name: letsencrypt-prod
    kind: ClusterIssuer
  commonName: wisecow.example.com
  dnsNames:
  - wisecow.example.com
```

Final Notes

Replace `<your-docker-registry>`

Problem Solution 2

Let’s tackle two objectives from the list: System Health Monitoring Script And Application Health Checker.Below are the implementation for both objectives using python.

1. System Health Monitoring Script

This Script will monitor CPU usage,memory usage disk space and running processes.if any of these metrics exceed predefined thresholds, it will log an alert to the console.

```
import psutil
import logging
import time

# Configure logging
logging.basicConfig(filename='system_health.log',
level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')

# Thresholds
CPU_THRESHOLD = 80  # in percent
MEMORY_THRESHOLD = 80  # in percent
DISK_THRESHOLD = 80  # in percent

def check_system_health():
   # Check CPU usage
   cpu_usage = psutil.cpu_percent(interval=1)
   if cpu_usage > CPU_THRESHOLD:
      logging.warning(f'High CPU usage detected:
{cpu_usage}%')
      print(f'High CPU usage detected: {cpu_usage}%')

   # Check memory usage
```

```
    memory_info = psutil.virtual_memory()
    memory_usage = memory_info.percent
    if memory_usage > MEMORY_THRESHOLD:
        logging.warning(f'High memory usage detected:
{memory_usage}%')
        print(f'High memory usage detected: {memory_usage}%')

    # Check disk usage
    disk_info = psutil.disk_usage('/')
    disk_usage = disk_info.percent
    if disk_usage > DISK_THRESHOLD:
        logging.warning(f'High disk usage detected:
{disk_usage}%')
        print(f'High disk usage detected: {disk_usage}%')

    # Check running processes
    processes = len(psutil.pids())
    logging.info(f'Number of running processes: {processes}')
    print(f'Number of running processes: {processes}')

if __name__ == "__main__":
    while True:
        check_system_health()
        time.sleep(60)  # Check every minute
```

2. Application Health Checker

This script will check the uptime of a specified application by sending an HTTP request ans checking the response status code.

```
import requests

def check_application_health(url):
    try:
        response = requests.get(url)
        if response.status_code == 200:
            print(f'The application at {url} is UP (Status Code:
{response.status_code})')
        else:
            print(f'The application at {url} is DOWN (Status Code:
{response.status_code})')
    except requests.exceptions.RequestException as e:
        print(f'The application at {url} is DOWN (Error: {e})')

if __name__ == "__main__":
    app_url = 'http://your-application-url.com'  # Replace with
your application's URL
    check_application_health(app_url)
```

How to use the Scripts

1.System Health Monitoring Script:

1. Save the first script as ‘system_health_monitor.py’.
2. Run the script using python:’python system_health_monitor.py’.
3. It will log system health metrics every minute and alert you if any thresholds are exceeded.

2.Application Health Checker:

1. Save this second script as ‘ app_health_checker.py’.
2.Replace ‘http://your application-url.com’ with the actual URL of your application.
3.Run the script using python :’python app_health_checker.py’.
4. It will check the application’s health and print whether it is up or down.

Requirements

Make sure you have the ‘p sutil’ and ‘requests’ libraries installed. You can install them using pip:

pip install psutil requests

These scripts provide a basic implementation of the specified objectives. You can enhance them further by adding more features, such as sending email alerts or integrating with monitoring tools.