Implementing Proactive Monitoring for Continuous Availability

PHASE 2- SOLUTION ARCHITECTURE

College Name: Shetty Institute of Technology Kalaburagi.

Group Members:

Name: Vinay S Buddhi
CAN ID Number: CAN_33269330
WORK: Application Requirements

Name: Sanni Kumar
CAN ID Number: CAN_33269278

Name: SOMESH HOTKAR
CAN ID Number: CAN_33259759

Name: Sneha Mnakoji
CAN ID Number: CAN_34000107

SOLUTION ARCHITECTURE

Building upon the Phase 1 design of MicroGuard, the phase 2 implementation focuses on enhancing the proactive monitoring system's scalability, efficiency, and reliability through a robust deployment architecture. Leveraging IBM Cloud Kubernetes Service and IBM Cloud Container Registry, the system will ensure continuous integration and deployment for a microservices-based environment.

1. **Create the main project folder:**

```
mkdir microguard-monitoring
cd microguard-monitoring
```

2. **Create the security-tools folder for vulnerability scanning scripts:**

```
mkdir public
```

3**: Create subfolders for CSS and JS under public**

```
mkdir public\css
mkdir public\js
```

4**: Create the necessary files in the public folder**

```
echo. > public\css\style.css
echo. > public\js\monitoring.js
echo. > public\dashboard.html
```

5: Create the server folder and its subfolders

mkdir server

mkdir server\controllers

mkdir server\models

mkdir server\routes

6: Create the necessary files in the server folder

echo. > server\controllers\metricsController.js

echo. > server\controllers\alertController.js

echo. > server\models\metricsModel.js

echo. > server\models\alertModel.js

echo. > server\routes\monitoringRoutes.js

echo. > server\routes\alertRoutes.js

echo. > server\server.js

Directory Structure

```
microguard-monitoring/
├── public/
│   ├── css/
│   │   └── style.css
│   ├── js/
│   │   └── monitoring.js
│   └── dashboard.html
├── server/
│   ├── controllers/
│   │   ├── metricsController.js
│   │   └── alertController.js
│   ├── models/
│   │   ├── metricsModel.js
│   │   └── alertModel.js
│   ├── routes/
│   │   ├── monitoringRoutes.js
│   │   └── alertRoutes.js
│   └── server.js
├── k8s/
│   ├── deployment.yaml
│   ├── service.yaml
│   └── configmap.yaml
├── Dockerfile
├── README.md
└── package.json
```

VERSION CONTROL SETUP

1. **Initialize Git in the project:**

git init

2. Create a .gitignore file to exclude unnecessary files:

echo node_modules/ > .gitignore

echo .env >> .gitignore

3. Commit the initial setup:

git add .

git commit -m "Initial setup for proactive monitoring project"

4 .Push to GitHub:

git remote add origin <repository_url>

git push -u origin master

CI/CD PIPELINE DESIGN AND IMPLEMENTATION

To automate the build, test, and deployment processes, we will design a CI/CD pipeline using Jenkins.

1. Jenkins Setup

We will use Jenkins to automate the build, test, and deployment of the MicroGuard system.

1. Install Jenkins and necessary plugins:
   - Docker
   - Git
   - Kubernetes CLI

2. Create a Jenkins Pipeline using the following Jenkinsfile:

```
pipeline {
    agent any
    environment {
        DOCKER_IMAGE = 'microguard-app'
        REGISTRY_URL = '<IBM_Container_Registry_URL>'
        CLUSTER_NAME = '<CLUSTER_NAME>'
    }
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/<username>/microguard-monitoring.git'
            }
        }
        stage('Build Docker Image') {
            steps {
                sh 'docker build -t $REGISTRY_URL/$DOCKER_IMAGE .'
            }
        }
        stage('Push to IBM Container Registry') {
            steps {
                sh 'docker push $REGISTRY_URL/$DOCKER_IMAGE'
            }
        }
```

```
        stage('Deploy to Kubernetes') {
            steps {
                sh '''
                ibmcloud login --apikey <API_KEY> -r <REGION> -g <RESOURCE_GROUP>
                ibmcloud ks cluster config --cluster $CLUSTER_NAME
                kubectl apply -f k8s/deployment.yaml
                '''
            }
        }
    }
    post {
        success {
            echo 'Pipeline executed successfully.'
        }
        failure {
            echo 'Pipeline failed. Check the logs.'
        }
    }
}
```

FUTURE PLAN

1. Advanced Monitoring:

Integratoe Prometheus and Grafana for comprehensive real-time analytics.

2. Auto-Remediation:

Implement machine learning models for automated issue resolution.

3. Distributed Monitoring:

Extend the system to monitor across multi-cloud environments.

4. DevSecOps Integration:

Secure the CI/CD pipeline with vulnerability scanning and image signing.