Demo: Automated Monitoring and Proactive Management Using Generative AI

Steps:

Step 1: Set Up a Node.js Application

Objective: Deploy a simple Node.js application that will be monitored.

Instructions:

1. Initialize the Node.js project:

   ```bash
   mkdir monitoring_demo
   cd monitoring_demo
   npm init -y
   ```

2. Install Express.js:

   ```bash
   npm install express
   ```

3. Create a basic `app.js` file:

   ```javascript
   const express = require('express');
   const app = express();
   ```

```
  const port = 3000;

  app.get('/', (req, res) => {
    res.send('Hello, Monitoring World!');
  });

  app.listen(port, () => {
    console.log(`Server running at http://localhost:${port}`);
  });
```

4. Run the application:

```bash
  node app.js
```

Step 2: Automated Monitoring Setup Using Generative AI (Step 6.01)

Objective: Use GitHub Copilot to configure Prometheus and Grafana for automated monitoring of the Node.js application.

Instructions:

1. Set up Prometheus:
   - Download Prometheus from the [official website](https://prometheus.io/download/).
   - Unzip and place the files in a convenient location.

2. Create a `prometheus.yml` configuration file:

   ```bash
   touch prometheus.yml
   ```

3. Prompt GitHub Copilot to generate the Prometheus configuration:

   - Open the `prometheus.yml` file and add the following comment to prompt GitHub Copilot:

     ```yaml
      Configure Prometheus to monitor a Node.js application
     ```

4. Let GitHub Copilot generate the Prometheus configuration:

   - Copilot will suggest:

     ```yaml
     global:
       scrape_interval: 15s

     scrape_configs:
       - job_name: 'node_app'
         static_configs:
           - targets: ['localhost:3000']
     ```

5. Run Prometheus:

   - Start Prometheus with the generated configuration:

     ```bash
     ./prometheus --config.file=prometheus.yml
     ```

6. Set up Grafana:

   - Download Grafana from the [official website](https://grafana.com/grafana/download).

- Install and start Grafana, then configure a Prometheus data source in Grafana's web interface by pointing it to `http://localhost:9090`.

7. Visualize the Node.js Application Metrics:

   - Create a simple Grafana dashboard to visualize metrics like response times, CPU usage, and memory usage.

Step 3: Proactive Management with AI-Driven Insights (Step 6.02)

Objective: Use Generative AI for anomaly detection and preventive actions using the integrated monitoring tools.

Instructions:

1. Write a Comment to Prompt GitHub Copilot for Anomaly Detection Setup:

   - In your Grafana dashboard configuration, write a comment to configure alerts for anomalies (e.g., high memory usage, CPU spikes):

```yaml
 Configure Grafana alerts for memory usage anomalies and CPU spikes
```

2. Let GitHub Copilot Generate the Alert Configuration:

   - Copilot will suggest something like this:

```yaml
alerting:
  alert_rules:
    - alert: HighMemoryUsage
      expr: process_resident_memory_bytes > 500000000
      for: 5m
      labels:
        severity: critical
      annotations:
        summary: "High memory usage detected"
        description: "The process is using over 500MB of memory."

    - alert: HighCPUUsage
      expr: process_cpu_seconds_total > 90
      for: 5m
      labels:
        severity: critical
      annotations:
        summary: "High CPU usage detected"
        description: "The CPU usage is above 90% for the past 5 minutes."
```

3. Enable Proactive Alerts:

   - Set up Grafana to send alerts when anomalies are detected, e.g., by configuring email or Slack notifications for critical alerts.

4. Test the Alerts:

   - Simulate high memory or CPU usage in the Node.js app by adding an artificial memory leak:

```javascript
app.get('/memory-leak', (req, res) => {
  let arr = [];
  for (let i = 0; i < 1e6; i++) {
    arr.push(i);
  }
  res.send('Memory leak simulated');
});
```

  - Check if Prometheus detects the memory spike and Grafana triggers an alert.

Step 4: Ensuring High Availability and Fault Tolerance (Step 6.03)

Objective: Apply Generative AI to optimize redundancy and mitigate downtime for high availability.

Instructions:

1. Add Load Balancing with NGINX:
   - Use NGINX to simulate load balancing for high availability.

2. Create an NGINX Configuration File:
   - In the root of your project, create an `nginx.conf` file:

    ```bash
    touch nginx.conf
    ```

3. Prompt GitHub Copilot to Generate a Load Balancing Configuration:
   - Write a comment in the `nginx.conf` file:

    ```bash
     Configure NGINX for load balancing between two Node.js instances
    ```

4. Let GitHub Copilot Generate the Load Balancer Config:
   - Copilot will generate something like this:

```nginx
upstream node_app {
    server localhost:3000;
    server localhost:3001;
}

server {
    listen 80;

    location / {
        proxy_pass http://node_app;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

5. Run Multiple Instances of Node.js:

- Start two instances of the Node.js app on different ports:

```bash
PORT=3000 node app.js
PORT=3001 node app.js
```

6. Run NGINX:

- Start NGINX with the generated configuration, ensuring it balances the load between the two Node.js instances for high availability.

7. Test Fault Tolerance:

- Stop one instance of the Node.js app and observe how NGINX automatically directs traffic to the remaining instance, demonstrating fault tolerance.
