Part 7 — All-in-One: System Monitoring Lab for Microservices

"Students are required to work in pairs for this lab."

Overview

This lab combines the content from Parts 1-6 and applies it to a microservices architecture. It focuses on deploying and monitoring microservices in a Kubernetes environment. Students will design and implement a simple multi-service application, instrument it with metrics, health checks, logging, and tracing, and deploy it using containers. The lab also covers integrating the system with Prometheus for metrics collection, Grafana for visualization, and configuring alerting rules to detect anomalies. An advanced component includes implementing a CI/CD pipeline to automate the build, test, and deployment process.

Learning Objectives

By the end of this lab, you will be able to:

1. Design and deploy a small microservices system on Kubernetes.
2. Instrument services with metrics, health checks, and structured logs.
3. Configure Prometheus to scrape application and infrastructure metrics.
4. Write useful PromQL queries and create Grafana dashboards.
5. Implement alerting via Alertmanager and test alert workflows.
6. Monitor containers and Kubernetes objects (pods, nodes, deployments).
7. Apply monitoring patterns for microservices (health checks, circuit breakers, request tracing, SLO/SLI basics).
8. Troubleshoot incidents using dashboards, metrics and traces.

Prerequisites

- Basic knowledge of Docker and Kubernetes concepts.
- A machine with kubectl, docker (or podman), and either minikube, kind, or a cloud Kubernetes cluster available.
- helm installed (v3+).
- git, curl, and a code editor.

Note: this lab will include commands and YAML you can paste into your terminal. Adjust namespaces and resource names as needed.

Lab topology (sample system)

The microservices system to be built and monitored is as follows.

- **api-gateway** - routes requests to backend services (simple Nginx or Node gateway).
- **order-service** - processes orders, emits business metrics and events.
- **inventory-service** - maintains inventory counts and exposes metrics.
- **frontend** - simple UI to generate traffic (can be a static app or an HTTP request generator).
- **user-service** - manages users and exposes HTTP endpoints and metrics (Optional).
- **payment-service** - simulates payment processing (Optional).

Support services:

- **Prometheus + Alertmanager** - metrics collection and alerting.
- **Grafana** - visualization and dashboarding.
- **kube-state-metrics** and **node-exporter** - Kubernetes and node metrics.
- **Loki** (or EFK) - aggregated logs (optional but recommended).
- **Jaeger** - distributed tracing (optional but recommended).

All services should run in Kubernetes objects (Deployments, Services) and expose /metrics (Prometheus) and /health endpoints.

Part 1 - Environment setup

- Initialize and start the Kubernetes environment.
- Enhance the cluster by enabling necessary add-ons to support resource monitoring.
- Configure external access management for services within the cluster.
- Organize the cluster resources using appropriate namespaces.
- Deploy a monitoring solution using a package management tool (Monitoring Stack).
- Verify the successful deployment of monitoring components.
- Access and review credentials for the visualization dashboard.
- Identify and describe the key components included in the monitoring stack.

Part 2 - Sample microservices deployment (Deploy Core Services and Instrumentation)

- Design and prepare the architecture for a simple demonstration application.
    - Include microservices that call each other using the circuit breaker pattern (e.g., order service calls inventory service).

    - Integrate an API gateway (NGINX Ingress Controller) for external access.
- Apply instrumentation practices to each microservice, including metrics, health checks, logging, and distributed tracing (Jaeger/OpenTelemetry).
- For each microservice, define basic and advanced metrics covering all four Prometheus metric types: Counters, Gauges, Histograms, and Summaries.
- Develop and implement the core microservices following the instrumentation guidelines.
- Test the functionality of the developed services to ensure correctness before deployment.
- Prepare container build configurations for each microservice.
- Implement a CI/CD pipeline (e.g., using GitHub Actions) to automate the build, test, and deployment of each microservice.
- Verify that all deployed services are correctly running and accessible.
- Perform basic API interactions to validate service communication and functionality (e.g., through frontend UI or Postman).
- Monitor the CI/CD pipeline logs to ensure successful builds, tests, and deployments for each microservice.

Part 3 - Prometheus Scraping Configuration (Connect Services to Prometheus)

Scraping: Prometheus periodically sends HTTP GET requests to the /metrics endpoint.

Metrics format: Exposed in plain text using Prometheus exposition format (e.g., counters, gauges, histograms).

Target discovery: Automatically done through Kubernetes API.

- Configure the monitoring system to collect metrics from the deployed microservices.
- Define or adjust the scraping configuration to include the newly deployed services.
- Ensure that each service exposes a metrics endpoint compatible with the monitoring system.
- Organize the scraping setup to target services within the appropriate namespace or environment.
- Validate that the monitoring tool successfully detects and gathers metrics from all relevant endpoints.
- Explore the collected metrics and verify their visibility within the monitoring dashboard.
- Investigate how label configuration and service discovery influence metric collection.

Part 4 - Visualization with Grafana

- Access the visualization platform included in the monitoring stack (Prometheus and Grafana).
- Connect the Grafana to the metrics data source.
- Explore the available dashboards and identify key visual components related to cluster and service performance.
- Create or customize dashboards to visualize metrics from the deployed microservices.
- Add and organize panels to display indicators such as resource usage, request rates, and error counts.
- Experiment with different visualization types (e.g., graphs, gauges, tables) to represent monitoring data effectively.
- Analyze the displayed metrics to interpret the health and behavior of the system.

- Save the configured dashboards.

Part 5: Configure and Manage Alerts.

- Configure the alerting component (Alertmanager) of the monitoring stack to track system and service health conditions.
- Define alerting rules based on the defined metrics. Set appropriate thresholds and conditions that trigger alerts when anomalies occur.
- Integrate the alerting system with the visualization platform for centralized monitoring (Alerts in Grafana).
- Explore how alert notifications can be routed to different channels (e.g., email, Slack, or other integrations).
- Test the alerting setup by simulating conditions that should trigger notifications.

Lab Deliverables – System Monitoring with Kubernetes and Microservices

Students are required to submit a detailed technical report (PDF) documenting the step-by-step execution of this lab. **The report should include each command in the correct order, accompanied by a screenshot of the results and a brief description (These details are mandatory).**

Submission Deadline (No delays will be tolerated): 19-11-2025

---------- **Good luck** ----------