**Observability and Monitoring of Microservices**

Trong kiến trúc phần mềm hiện đại, microservices là xu hướng phổ biến nhờ khả năng mở rộng linh hoạt và quản lý độc lập từng thành phần. Tuy nhiên, đi cùng với lợi ích là những thách thức trong việc giám sát hoạt động và phát hiện sự cố khi hàng loạt dịch vụ nhỏ vận hành cùng lúc. Để đảm bảo hệ thống hoạt động ổn định, hiệu quả và dễ bảo trì, các kỹ thuật observability (khả năng quan sát) và monitoring (giám sát) trở nên vô cùng quan trọng.

1. **Observability (Quan sát hệ thống)**

- **Khái niệm**: Khả năng hiểu được trạng thái nội bộ của hệ thống thông qua các dữ liệu đầu ra như logs, metrics và traces.
- **3 trụ cột chính**:
    - **Metrics**: Số liệu định lượng như CPU, RAM, thời gian phản hồi.
    - **Logs**: Bản ghi chi tiết các sự kiện, hữu ích khi debug.
    - **Traces**: Theo dõi luồng đi của request qua các microservices.
- **Mục tiêu**: Giúp **hiểu được hệ thống hoạt động ra sao** và phát hiện sớm lỗi, bottlenecks, bất thường.
- **Cách tiếp cận**: **Proactive (chủ động)** – phát hiện và khắc phục sự cố trước khi chúng gây ra ảnh hưởng lớn.

2. **Monitoring (Giám sát hệ thống)**

- **Khái niệm**: Giám sát dữ liệu telemetry (metrics, logs, traces) để phát hiện sự cố và gửi cảnh báo.
- **Chức năng chính**:
    - Thiết lập **dashboards**, **alerts**, **notifications**.

- Theo dõi **health**, hiệu suất các microservice.

- **Ví dụ**: Gửi cảnh báo khi CPU > 80%, scale-out thêm instance để tránh quá tải.
- **Cách tiếp cận**: **Reactive (phản ứng)** – xử lý sau khi vấn đề xảy ra.

3. **Phân biệt Observability và Monitoring**

Monitoring là quá trình thu thập các số liệu (metrics), logs và trạng thái hệ thống để hiển thị, cảnh báo khi có sự cố xảy ra. Nó trả lời câu hỏi "khi nào" hệ thống gặp sự cố. Trong khi đó, observability là khả năng hiểu được bên trong hệ thống thông qua dữ liệu đầu ra, giúp phân tích nguyên nhân gốc rễ của vấn đề, từ đó hỗ trợ xử lý triệt để và nhanh chóng hơn.

4. **Logging là gì?**

- Là các bản ghi (record) của những sự kiện xảy ra trong ứng dụng.
- Gồm **timestamp**, nội dung sự kiện, **context** (như thread, user, tenant...).
- Giúp **debug**, **truy vết lỗi**, và **phân tích hệ thống**.

5. **So sánh logging giữa Monolith và Microservices**

| **Monolith** | **Microservices** |
|---|---|
| Một codebase → log tập trung | Nhiều service → log phân tán |
| Dễ tìm log, debug đơn giản | Khó tra cứu log từ nhiều container |
| Log lưu tại 1 nơi duy nhất | Log nằm rải rác khắp hệ thống |

6. **Triển khai Logging tập trung**

Một hệ thống microservices cần có khả năng thu thập log tập trung để tiện theo dõi và phân tích. Bộ công cụ được sử dụng trong bài học gồm:

- Promtail: thu thập log từ các container hoặc máy chủ,
- Loki: lưu trữ và hỗ trợ truy vấn log hiệu quả,

- Grafana: hiển thị log dưới dạng bảng, biểu đồ dễ hiểu.

Việc tích hợp ba công cụ này cho phép xây dựng hệ thống logging tập trung, hỗ trợ truy xuất log theo thời gian, từ đó phát hiện và xử lý lỗi nhanh chóng hơn trong môi trường nhiều service.

7. **Giám sát hệ thống bằng Metrics**

Metrics là những chỉ số định lượng phản ánh tình trạng và hiệu năng của hệ thống. Các công cụ được tích hợp gồm:

- Micrometer: thư viện tích hợp vào Spring Boot để thu thập dữ liệu như số lượng request, độ trễ, CPU usage…
- Prometheus: lưu trữ và xử lý metrics theo thời gian.
- Grafana: hiển thị các biểu đồ, bảng số liệu và dashboard tùy chỉnh.

Người học cũng được hướng dẫn tạo alerts (cảnh báo) dựa trên các điều kiện cụ thể, nhằm gửi thông báo qua email hoặc các nền tảng như Slack khi hệ thống vượt ngưỡng an toàn.

8. **Distributed Tracing – Theo dõi hành trình của request**

Trong microservices, một request có thể đi qua nhiều service trước khi hoàn tất. Distributed tracing giúp theo dõi đầy đủ hành trình đó. Bài học sử dụng:

- OpenTelemetry: chuẩn thu thập dữ liệu trace,
- Tempo: lưu trữ trace,
- Grafana: hiển thị trace theo dạng thời gian, cây gọi lồng nhau (call tree).

9. **Metrics & Monitoring với Micrometer, Prometheus & Grafana**

- Micrometer: Thư viện thu thập metrics trong Spring Boot.
- Prometheus: Công cụ thu thập metrics, pull dữ liệu từ endpoint /actuator/prometheus.

- Grafana: Dashboard để hiển thị metrics.
- Demo - Prometheus inside microservices:
    - Cấu hình Prometheus để lấy dữ liệu từ từng service.
    - Hiển thị các chỉ số như HTTP requests, memory, heap, GC...
- Demo - Grafana inbuilt & custom Dashboards:
    - Sử dụng dashboard có sẵn của Grafana để giám sát hệ thống.
    - Tạo dashboard tùy chỉnh với biểu đồ riêng, lọc theo service hoặc chỉ số cụ thể.
- Demo - Create Alerts & Send Notifications:
    - Cấu hình alert trong Grafana khi vượt ngưỡng CPU, lỗi 500, v.v.
    - Thiết lập gửi thông báo qua Email/Slack qua 2 cách:
        - Approach 1: Notification channel truyền thống.
        - Approach 2: Sử dụng Alert Rules + Contact Points.

**10. Distributed Tracing với OpenTelemetry, Grafana Tempo**

- OpenTelemetry: Thư viện thu thập trace trong các microservices.
- Tempo: Công cụ lưu trữ traces, tích hợp mượt mà với Grafana.
- Grafana: Truy xuất và phân tích trace chi tiết.
- Demo - Implement OpenTelemetry & Tracing using Grafana:
    - Cấu hình SDK OpenTelemetry trong Spring Boot.
    - Gửi trace tới Tempo và hiển thị hành trình của một request đi qua nhiều services.
    - Điều tra latency, bottlenecks.

- Tính năng điều hướng từ logs sang traces:
    - Từ một dòng log trong Loki, có thể truy xuất sang trace tương ứng trong Tempo, tạo mối liên kết chặt chẽ giữa log và trace.