Final Project: Monitoring & Observability with Prometheus and Grafana

Monitoring project queries

Error rate:
rate(http_requests_errors_total[5m]) / rate(http_requests_total[5m]).

Latency quantile:
histogram_quantile(0.95,rate(http_request_duration_seconds_bucket[5m])).

Used memory:
(node_memory_MemTotal_bytes - node_memory_MemAvailable_bytes) / node_memory_MemTotal_bytes * 100

Chaos test & reporting

For this test, I generated a planned CPU load using the command "stress --cpu 3 --timeout 120s" to illustrate the server's load profile.
In a real-world scenario where such CPU pressure is observed, a DevOps engineer should:

- Investigate the source of the load (application process, misconfiguration, or external requests).
- Check system metrics and logs to confirm whether the spike is expected or anomalous.

- Set alerts and thresholds in monitoring tools to detect and respond faster in future cases.

Report

System Architecture Diagram:
The system consists of servers monitored with Prometheus for metrics collection, Alertmanager for handling alerts, and Grafana for dashboards. Data flows from Prometheus to Grafana, and alerts are sent via Alertmanager to email.

Key PromQL Queries and Their Purpose

- **Error rate:**
  rate of HTTP request errors over total requests in the last 5 minutes. This monitors the rate of HTTP request errors, enabling real-time detection of service issues.

- **Latency quantile (95th percentile):**
  95th percentile of request duration in seconds over the last 5 minutes. This measures the highest response times to ensure the system remains fast and stable.

- **Used memory:**
  percentage of memory in use, calculated as total memory minus available memory divided by total memory times 100. This helps identify resource pressure or shortages.

Reflection

The monitoring setup allows us to detect unusual resource loads and slow response times in real time. In the case of high CPU usage, memory pressure, or increased HTTP errors, a DevOps engineer should:

- Check which processes consume the most resources and consider optimization.

- Consider load balancing across servers or increasing available resources.

- Ensure the system continues to run stably and address issues according to priority.