Contents

Scenario: Monitor and Auto-Scale Microservices

Describe your monitoring, alerting, and auto-scaling strategy for a microservices architecture.

Summary

Instrument with Prometheus/Grafana. Set autoscaling policies based on RPS (Requests per Second) or CPU. Use HPA (Horizontal Pod Autoscaler)/VPA (Vertical Pod Autoscaler). Include circuit breakers and throttling.

Monitoring & Instrumentation

- **Metrics Collection:**
  - Use Prometheus to scrape metrics from all microservices (e.g., latency, error rate, RPS, CPU/memory usage).
  - Expose custom application metrics via endpoints (e.g., /metrics).
- **Visualization:**
  - Use Grafana to create dashboards for real-time monitoring of service health and performance.
- **Distributed Tracing:**
  - Integrate tools like Jaeger or OpenTelemetry to trace requests across services and identify bottlenecks.
  - Consider eBPF-based observability tools (e.g., Cilium, Pixie, BPFTrace) for low-overhead, kernel-level tracing and deep system visibility.

Alerting

- **Alert Policies:**
  - Set up alerts for SLO/SLA violations (e.g., high error rate, increased latency, resource exhaustion).
  - Use alerting rules in Prometheus or integrate with PagerDuty/Slack for incident response.
- **Runbooks:**
  - Document standard operating procedures for common alerts to speed up incident resolution.

Auto-Scaling Strategies

- **Horizontal Pod Autoscaler (HPA):**
  - Automatically increases or decreases the number of pods based on observed metrics (CPU, memory, RPS).
  - Configure thresholds and cool-down periods to avoid flapping.
- **Vertical Pod Autoscaler (VPA):**
  - Adjusts CPU and memory requests/limits for pods based on usage patterns.
  - Use with caution—may cause pod restarts.
- **Custom Metrics:**
  - Scale on business metrics (e.g., queue length, request latency) for more accurate scaling.

Resilience: Circuit Breakers & Throttling

- **Circuit Breakers:**
  - Prevent cascading failures by stopping requests to unhealthy services.
  - Use libraries like Hystrix or built-in features in service meshes (e.g., Istio).
- **Throttling & Rate Limiting:**
  - Protect services from overload by limiting the number of requests per client or per service.
  - Implement at API gateway or service mesh layer.

Key Metrics to Monitor

- Request rate (RPS)
- Error rate (5xx, 4xx)
- Latency (p99, p95)
- CPU and memory utilization
- Pod/container restarts
- Autoscaler activity (scale up/down events)