

How do we handle Kafka lag spikes?

Context: Aurelius Systems (Aurelius Systems) backend runs on Spring Boot microservices with Kafka, Redis, MySQL, AWS ALB, and Prometheus/Grafana.

Detection: Issues are detected using alerts, dashboards, and correlated logs via requestId and traceId.

Root Cause Analysis: The root cause involved a mix of traffic spikes, resource saturation, configuration limits, and slow downstream calls.

Debugging Steps:

- Analyze CPU, memory, latency, TPS metrics
- Inspect logs and distributed traces
- Check thread dumps, DB pools, Kafka lag
- Compare with historical baselines

Fixes Applied:

- Code-level optimizations
- Configuration tuning
- Scaling policies
- Timeouts, retries, circuit breakers

Prevention:

- Load testing
- Better alerting
- Capacity planning
- Clear runbooks and SOPs

Context: Aurelius Systems (Aurelius Systems) backend runs on Spring Boot microservices with Kafka, Redis, MySQL, AWS ALB, and Prometheus/Grafana.

Detection: Issues are detected using alerts, dashboards, and correlated logs via requestId and traceId.

Root Cause Analysis: The root cause involved a mix of traffic spikes, resource saturation, configuration limits, and slow downstream calls.

Debugging Steps:

- Analyze CPU, memory, latency, TPS metrics
- Inspect logs and distributed traces
- Check thread dumps, DB pools, Kafka lag
- Compare with historical baselines

Fixes Applied:

- Code-level optimizations
- Configuration tuning
- Scaling policies
- Timeouts, retries, circuit breakers

Prevention:

- Load testing
- Better alerting
- Capacity planning
- Clear runbooks and SOPs

Context: Aurelius Systems (Aurelius Systems) backend runs on Spring Boot microservices with Kafka, Redis, MySQL, AWS ALB, and Prometheus/Grafana.

Detection: Issues are detected using alerts, dashboards, and correlated logs via requestId and traceId.

Root Cause Analysis: The root cause involved a mix of traffic spikes, resource saturation, configuration limits, and slow downstream calls.

Debugging Steps:

- Analyze CPU, memory, latency, TPS metrics
- Inspect logs and distributed traces
- Check thread dumps, DB pools, Kafka lag
- Compare with historical baselines

Fixes Applied:

- Code-level optimizations
- Configuration tuning
- Scaling policies
- Timeouts, retries, circuit breakers

Prevention:

- Load testing
- Better alerting
- Capacity planning
- Clear runbooks and SOPs

Context: Aurelius Systems (Aurelius Systems) backend runs on Spring Boot microservices with Kafka, Redis, MySQL, AWS ALB, and Prometheus/Grafana.

Detection: Issues are detected using alerts, dashboards, and correlated logs via requestId and traceId.

Root Cause Analysis: The root cause involved a mix of traffic spikes, resource saturation, configuration limits, and slow downstream calls.

Debugging Steps:

- Analyze CPU, memory, latency, TPS metrics
- Inspect logs and distributed traces
- Check thread dumps, DB pools, Kafka lag
- Compare with historical baselines

Fixes Applied:

- Code-level optimizations
- Configuration tuning
- Scaling policies
- Timeouts, retries, circuit breakers

Prevention:

- Load testing
- Better alerting
- Capacity planning
- Clear runbooks and SOPs

Context: Aurelius Systems (Aurelius Systems) backend runs on Spring Boot microservices with Kafka, Redis, MySQL, AWS ALB, and Prometheus/Grafana.

Detection: Issues are detected using alerts, dashboards, and correlated logs via requestId and traceId.

Root Cause Analysis: The root cause involved a mix of traffic spikes, resource saturation, configuration limits, and slow downstream calls.

Debugging Steps:

- Analyze CPU, memory, latency, TPS metrics
- Inspect logs and distributed traces
- Check thread dumps, DB pools, Kafka lag
- Compare with historical baselines

Fixes Applied:

- Code-level optimizations
- Configuration tuning
- Scaling policies
- Timeouts, retries, circuit breakers

Prevention:

- Load testing
- Better alerting
- Capacity planning
- Clear runbooks and SOPs

Context: Aurelius Systems (Aurelius Systems) backend runs on Spring Boot microservices with Kafka, Redis, MySQL, AWS ALB, and Prometheus/Grafana.

Detection: Issues are detected using alerts, dashboards, and correlated logs via requestId and traceId.

Root Cause Analysis: The root cause involved a mix of traffic spikes, resource saturation, configuration limits, and slow downstream calls.

Debugging Steps:

- Analyze CPU, memory, latency, TPS metrics
- Inspect logs and distributed traces
- Check thread dumps, DB pools, Kafka lag
- Compare with historical baselines

Fixes Applied:

- Code-level optimizations
- Configuration tuning
- Scaling policies
- Timeouts, retries, circuit breakers

Prevention:

- Load testing
- Better alerting
- Capacity planning
- Clear runbooks and SOPs

[illegible]

Compare with historical baselines Fixes Applied: - Code-level optimizations - Configuration tuning - Scaling policies - Timeouts, retries, circuit breakers Prevention: - Load testing - Better alerting - Capacity planning - Clear runbooks and SOPs