Kafka Incident Management Runbooks

Purpose and Function of Kafka

Apache Kafka is a distributed event streaming platform used for building real-time data pipelines and streaming applications. It is designed for high-throughput, fault tolerance, and scalability. Kafka allows producers to send messages (events) to topics, which are then consumed by consumers. It decouples data producers from consumers, enabling asynchronous communication, buffering, and fault-tolerant data delivery. Kafka is widely used for log aggregation, real-time analytics, event sourcing, and stream processing.

Kafka Failure Troubleshooting (Development & Production)

Troubleshooting Kafka failures involves structured steps: recognizing symptoms, diagnosing the root cause, performing immediate actions, and implementing remediation to prevent recurrence. Both development and production environments require distinct approaches due to differences in scale, architecture, monitoring, and criticality of services.

Why These Failure Modes Were Chosen

Development Failure Mode: Broker Unavailable or Unresponsive

In developer environments, Kafka is typically run via Docker or as an embedded service. The most frequent issues arise from Kafka not starting correctly, misconfigurations, or local resource issues such as disk space or port conflicts. This failure mode is simple but impactful, as it blocks development workflows, service integration tests, and local CI/CD feedback loops. Addressing this failure mode helps keep developers productive and reduces noise in development-related support requests.

Production Failure Mode: Partial or Total Broker Failure / Cluster Instability

In production environments, failures are more complex and can cascade across the Kafka cluster. This includes broker crashes, leader election issues, ISR (in-sync replica) shrinkage, and network partitions. These failures directly affect message delivery, service SLAs, and overall system stability. This mode was chosen because it reflects the most critical real-world impact and requires a structured SRE/DevOps response to avoid data loss, duplication, or ingestion delays.

Kafka Development Runbook

Environment

Development

Failure Mode

- Broker Unavailable or Unresponsive

Symptoms

- Kafka client errors: TimeoutException, Connection Refused
- CI/CD steps fail on Kafka communication
- Local Kafka (Docker/embedded) unresponsive
- Development services not producing or consuming messages

Diagnostics

- Check broker logs (Docker or local)
- netstat -tuln | grep 9092
- kafka-topics.sh --list --bootstrap-server localhost:9092
- df -h (check disk space)

Possible Causes

- Kafka not started or crashed on boot
- Misconfigured server.properties
- Port conflict or unavailable
- Zookeeper unavailable (if used)
- Docker volume corruption

Actions

- Restart Kafka or Docker container
- Fix server.properties issues
- Free up disk space
- Rebuild or prune Docker containers
- Start/restart Zookeeper

Remediation

- Add Docker health checks for Kafka
- Embed Kafka in test environments
- Improve dev bootstrap scripts
- Document developer setup prerequisites

Kafka Production Runbook

Environment

Production

Failure Mode

- Partial or Total Broker Failure / Cluster Instability

Symptoms

- Kafka client timeouts (produce/consume)

- Consumer group lag increase
- Under-replicated partitions
- Brokers marked as offline in metrics
- Errors: NotLeaderForPartitionException, TimeoutException
- Drop in ingestion throughput

Diagnostics

- Check broker health with CLI or metrics dashboard
- kafka-consumer-groups.sh --describe
- kafka-broker-api-versions.sh
- echo stat | nc localhost 2181 (ZooKeeper)
- Check disk space: df -h
- Broker and GC logs
- Network/firewall checks

Possible Causes

- Broker crashed due to OOM or corrupted log
- Disk full on broker node
- ZooKeeper unavailable or unstable
- Leader election stuck due to ISR issues
- Network partition
- Misconfigured listeners

Actions

- Restart affected brokers
- Clean corrupted logs if needed
- Reassign partitions
- Restore ZooKeeper health or failover
- Throttle producers
- Add brokers to scale

Remediation

- Enable autoscaling for Kafka nodes
- Disk space monitoring and alerting
- Replication factor ≥ 3, ISR monitoring
- Kafka self-healing configs (unclean.leader.election.enable: false)
- Enable producer retries and idempotence
- Regular partition rebalancing
- Transition to KRaft

Reliability Engineering Notes

- Use Prometheus + Grafana with Kafka Exporter
- Centralized log aggregation

- Circuit breakers, backpressure, DLQ
- Chaos testing and failover exercises
- Maintain up-to-date runbooks and drills