Comprehensive Study Material: Troubleshooting Confluent Kafka

Table of Contents

1. Introduction & Foundational Concepts

2. Broker-Level Troubleshooting

3. Consumer & Producer Troubleshooting

4. Network & Connectivity Issues

5. Security & Authentication Problems

6. Monitoring & Diagnostics

7. Performance & Tuning Issues

8. Common Error Codes & Solutions

9. Best Practices & Preventive Measures

1. Introduction & Foundational Concepts

1.1 Understanding Confluent Kafka Architecture

Confluent Kafka is an enterprise-grade Apache Kafka distribution with additional management tools and features. The core components include:

Brokers: Core Kafka servers that manage partitions and handle produce/consume requests

ZooKeeper/KRaft: Coordination layer (ZooKeeper is legacy; KRaft is the new mode)

Control Center: Web-based monitoring and management interface

Schema Registry: Manages data schemas for structured messages

Kafka Connect: Framework for integrating external systems

ksqlDB: SQL-based stream processing engine

1.2 Troubleshooting Mindset

Effective troubleshooting follows a systematic approach:

1. Identify: Recognize symptoms and anomalies
2. Isolate: Determine affected components (broker, topic, consumer, etc.)
3. Investigate: Analyze logs, metrics, and configurations
4. Remediate: Apply targeted fixes
5. Validate: Verify resolution and prevent recurrence

2. Broker-Level Troubleshooting

2.1 Broker Startup Issues

Problem: Broker fails to start

Common causes and solutions:

- JVM Configuration: Ensure sufficient heap memory is allocated
  - Set `KAFKA_HEAP_OPTS=-Xmx4G -Xms4G` (adjust based on available memory)
  - Default is often too low for production workloads

- Port Conflicts: Verify ports are available
  - Default listener port: 9092
  - Check: `netstat -tuln | grep 9092` (Linux/Mac) or `netstat -ano | findstr 9092` (Windows)

-Log Directory Permissions: Ensure the broker can write to log directories
  - Check: `ls -la /var/kafka/data`

  - Fix: `chmod -R 755 /var/kafka/data` and set proper ownership

- Corrupted State Files: ZooKeeper or KRaft state may be corrupted
  - Backup and remove state: `rm -rf /var/kafka/data/__uuid_registry_metadata.log`
  - Restart broker to recreate

Diagnostic Steps:

```
# Check broker logs
tail -f /var/log/kafka/server.log

# Verify JVM startup
ps aux | grep kafka

# Test broker connectivity
nc -zv localhost 9092
```

2.2 Broker Crashes & Recovery

Problem: Broker crashes unexpectedly

Investigation Checklist:

1. Review Error Logs
   - Look for OutOfMemoryError, SocketException, or IOException
   - Pattern: `grep -i "error\|exception" /var/log/kafka/server.log | tail -50`

2. Memory Issues

   - Monitor with: `jps -m` (Java processes with memory)

   - JMX metrics: `kafka.server:type=BrokerTopicMetrics,name=*`

   - If heap exhausted: increase `KAFKA_HEAP_OPTS` and `KAFKA_JVM_PERFORMANCE_OPTS`

3. Disk Space

   - Kafka cannot write logs if disk is full

   - Check: `df -h /var/kafka/data`

   - Solution: Delete old segments or expand disk

4. Network Saturation

   - Monitor: `netstat -s` for dropped packets

   - Check broker's `NetworkMetrics` via JMX

   - Adjust: `num.network.threads` and `num.io.threads`

Recovery Procedure:

- Single broker failure: Replication handles failover automatically
- If leader is affected: ISR (In-Sync Replicas) promote next replica
- Full cluster failure: Restart brokers in any order; leader election occurs

2.3 Partition & Replication Issues

Problem: Under-replicated partitions

This is a critical issue indicating data loss risk.

Symptoms:

- Control Center shows "Under Replicated Partitions"
- Metric: `kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions`

Causes & Solutions:

1. Broker Offline
   - Check: `bin/kafka-broker-api-versions.sh --bootstrap-server localhost:9092`
   - If broker unresponsive: check network, logs, disk space

2. Slow Broker (Replica Lag)
   - Metric: `kafka.server:type=ReplicaFetcherManager,name=MaxLag,clientId=Replica`
   - Solution: Increase `replica.lag.time.max.ms` or optimize broker resources

3. ISR (In-Sync Replicas) Shrinkage
   - Monitor: `kafka.server:type=ReplicaManager,name=ISRShrinks`
   - Root cause: Leader detects replica as stale
   - Recovery: Fix the slow/offline broker; ISR will expand when replica catches up

Diagnostic Commands:

```
# Describe topic with details
kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic my-topic

# Check replica log end offset
kafka-replica-verification.sh --broker-list localhost:9092 --topic-white-list ".*"
```

```
# Monitor ISR changes
kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --all-groups
```

2.4 Leader Election Problems

Problem: Leader election takes too long or fails

Key Factors:

1. Unclean Leader Election (`unclean.leader.election.enable=false` recommended)
   - If enabled, allows data loss
   - When no in-sync replicas available, picks newest replica (potentially lost data)
   - Keep disabled; fix underlying replication issues instead

2. Stuck Leader Elections
   - Cause: ZooKeeper/KRaft coordination delays
   - Check ZooKeeper health: `echo stat | nc localhost 2181`
   - Monitor: `kafka.server:type=KafkaRequestHandlerPool,name=AvgIdlePercent`

3. Broker Not Recovering State
   - Brokers need time to recover state on startup
   - Monitor: `kafka.server:type=ReplicaManager,name=LeaderCount`
   - Allow adequate startup time before declaring failure

3. Consumer & Producer Troubleshooting

3.1 Producer Issues

Problem: Producer fails to send messages

Common Error Codes:

1. LEADER_NOT_AVAILABLE
   - Meaning: No leader for partition
   - Solution: Wait for leader election; check broker health
   - Code: Implement exponential backoff retry logic

2. REQUEST_TIMED_OUT
   - Meaning: Broker didn't respond in time
   - Solutions:
     - Increase `request.timeout.ms` (default: 30s)
     - Increase `delivery.timeout.ms` (default: 2min)
     - Increase broker's `num.network.threads`

3. MESSAGE_TOO_LARGE
   - Meaning: Message exceeds max size
   - Check: `message.max.bytes` (broker) and `max.request.size` (producer)
   - Solution: Increase both or compress messages

4. NOT_LEADER_FOR_PARTITION
   - Meaning: Partition leader changed
   - Solution: Producer automatically updates metadata; may retry failed sends

Producer Configuration Tuning:

```properties
# Reliability
acks=all  # Wait for all replicas; safest but slower
retries=3
max.in.flight.requests.per.connection=1  # Ensures ordering with retries

# Performance
batch.size=16384  # Bytes; larger = better throughput
linger.ms=10  # Wait up to 10ms for batching
compression.type=snappy  # or lz4, gzip

# Timeouts
request.timeout.ms=30000
delivery.timeout.ms=120000
```

Debugging:

```
# Enable producer logs
export KAFKA_LOG4J_OPTS="-Dlog4j.configuration=file:log4j.properties"

# Monitor producer metrics via JMX
# Key metrics:
- kafka.producer:type=producer-metrics,client-id=*
- kafka.producer:type=producer-topic-metrics,client-id=*,topic=*
```

3.2 Consumer Lag Issues

Problem: Consumer group is far behind (high lag)

Understanding Consumer Lag:

- Lag: Difference between latest offset (partition end offset) and consumer's committed offset
- Formula: `Max Offset - Consumer Offset`
- Critical metric for monitoring data freshness

Root Causes:

1. Slow Consumer Processing
   - Check consumer processing time: `process.duration_ms`
   - Monitor: `poll()` frequency and handler performance
   - Solution: Optimize application logic or increase consumer instances

2. Consumer Crashes or Rebalancing
   - Rebalancing causes pause in message consumption
   - During rebalancing: no progress on lag
   - Symptom: Lag spikes correlated with consumer logs

3. Topic Message Rate > Consumer Rate
   - Producers sending faster than consumer can process
   - Solution: Scale consumers horizontally

4. Network Issues

   - Broker-to-consumer latency

   - Check: `fetch.wait.max.ms` (broker) and `fetch.min.bytes` (consumer)

Consumer Configuration Tuning:

```
properties
# Parallelism
max.poll.records=500  # Default 500; tune based on message size
fetch.min.bytes=1  # Minimum bytes before returning; default 1
fetch.max.wait.ms=500  # Max wait time; default 500ms

# Session management
session.timeout.ms=30000  # Default; increase if rebalancing frequently
heartbeat.interval.ms=10000  # Should be 1/3 of session.timeout.ms

# Offset commit
enable.auto.commit=true  # or false for manual control
auto.commit.interval.ms=5000  # Commit frequency

# Performance
max.partition.fetch.bytes=1048576  # Max bytes per partition; default 1MB
connections.max.idle.ms=540000  # Keep-alive
```

Monitoring Consumer Lag:

```
# Via Consumer Group Command
kafka-consumer-groups.sh --bootstrap-server localhost:9092 \
  --group my-consumer-group --describe

# Output columns:
TOPIC, PARTITION, CURRENT-OFFSET, LOG-END-OFFSET, LAG, CONSUMER-ID, HOST, CLIENT-ID

# Interpret:
- LAG = 0: Consumer caught up
- LAG > 0: Consumer behind
- CURRENT-OFFSET increases over time = consumer processing
```

Remediation Steps:

1. Identify bottleneck: Is it consumer or producer?
2. Increase consumers: Add instances up to partition count
3. Optimize processing: Profile application code
4. Monitor rebalancing: Minimize frequency and duration
5. Set up alerting: Alert when lag exceeds threshold

3.3 Consumer Group Rebalancing Issues

Problem: Excessive rebalancing causing service interruption

Rebalancing Triggers:

1. Consumer joins/leaves group
2. Consumer heartbeat missed

3. Partition count changes

4. Manual partition reassignment

Issues During Rebalancing:

- Stop-the-world: No message processing during rebalance
- Impact: Application unavailability, lag growth
- Duration: Typically 5-30 seconds; longer in large clusters

Solutions:

1. Extend Session Timeout
   - Increase `session.timeout.ms` (default: 30s) to 60-90s
   - Reduces rebalancing from temporary network glitches
   - Trade-off: Delayed detection of failed consumers

2. Stable Assignment Strategy
   - Use `partition.assignment.strategy=RoundRobin` or `StickyAssignor`
   - `CooperativeStickyAssignor`: Minimizes stop-the-world time (newer, recommended)

3. Graceful Shutdown

   python

```
# In consumer code
try:
    while True:
        records = consumer.poll(100)
        # process records
except KeyboardInterrupt:
    consumer.close()  # Graceful close triggers cooperative rebalance
```

4. Monitor Rebalancing Events
   - Metric: `kafka.consumer:type=consumer-coordinator-metrics,client-id=*`
   - Look for: `commit-latency-avg` and `assigned-partitions` changes

4. Network & Connectivity Issues

4.1 Connection Failures

Problem: Cannot connect to Kafka cluster

Diagnostic Checklist:

1. Verify Bootstrap Servers

```
  # Test DNS resolution
  nslookup kafka-broker-1.example.com

  # Test TCP connectivity
  telnet kafka-broker-1.example.com 9092
  or nc -zv kafka-broker-1.example.com 9092
```

2. Firewall Rules
   - Ensure ports are open: 9092 (plaintext), 9093 (SSL), 9094 (SASL)
   - Check: `sudo iptables -L -n` or cloud provider's security groups

3. Broker Listener Configuration

   - In `server.properties`:

```
    listeners=PLAINTEXT://0.0.0.0:9092
    advertised.listeners=PLAINTEXT://kafka-broker-1.example.com:9092
```

   - `listeners`: Internal binding
   - `advertised.listeners`: What clients use to connect back

4. Client Configuration

```
   bootstrap.servers=kafka-broker-1.example.com:9092,kafka-broker-2.example.com:9092
```

4.2 Metadata Refresh Issues

Problem: Broker list updates fail; clients stuck with stale metadata

Root Causes:

1. Advertised Listeners Misconfigured
   - Broker advertises unreachable address
   - Check: `kafka-broker-api-versions.sh --bootstrap-server localhost:9092`
   - Verify advertised addresses are reachable from client perspective

2. DNS Inconsistency
   - Broker hostname resolves differently on client vs. broker
   - Solution: Use IPs or consistent FQDN

3. Metadata Cache Stale
   - Client caches metadata; refreshes on errors or timeout
   - `metadata.max.age.ms`: 5min default
   - Force refresh: Retry on MetadataException

4.3 Inter-Broker Communication Issues

Problem: Brokers cannot communicate with each other

Symptoms:
- Replication lag
- Leader election delays
- Under-replicated partitions

Diagnosis:

1. Check Inter-Broker Protocol
   - `inter.broker.protocol.version` in `server.properties`
   - All brokers must support same version

2. Test Broker-to-Broker Connectivity

```
   # From broker A, test connectivity to broker B
   nc -zv broker-b.example.com 9092

   # Check broker logs for connection errors
   grep "Connection refused\|Network is unreachable" /var/log/kafka/server.log
```

3. Verify Replication Configuration

- `min.insync.replicas`: Should be < replication factor
- `default.replication.factor`: Ensure adequate replicas

5. Security & Authentication Problems

5.1 SSL/TLS Certificate Issues

Problem: SSL handshake failures

Common Errors:

1. CERTIFICATE_VERIFY_FAILED

```
javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException
```

- Cause: Client doesn't trust broker's certificate
- Solution:

```
     # Import broker certificate to client truststore
   keytool -import -alias kafka-broker -file broker-cert.pem \
     -keystore client-truststore.jks
```

2. HOSTNAME_MISMATCH

- Certificate CN or SAN doesn't match broker hostname
- Solution: Regenerate certificate with correct hostname

3. CERTIFICATE_EXPIRED

   - Check: `openssl x509 -in broker-cert.pem -noout -dates`

   - Renew certificate before expiration

Client Configuration for SSL:

```
properties

# Producer/Consumer

security.protocol=SSL

ssl.truststore.location=/path/to/truststore.jks

ssl.truststore.password=password

ssl.keystore.location=/path/to/keystore.jks  # For mTLS

ssl.keystore.password=password

ssl.key.password=password
```

5.2 SASL Authentication Issues

Problem: SASL authentication fails

Common Scenarios:

1. SASL/PLAIN (Username/Password)

```
   properties

   security.protocol=SASL_PLAINTEXT

   sasl.mechanism=PLAIN

   sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule \

     required username="user1" password="secret";
```

   - Verify credentials in broker's `jaas.config`

2. SASL/SCRAM
   - More secure than PLAIN

```
   # Create user in broker
   kafka-configs.sh --bootstrap-server localhost:9092 \
     --entity-type users --entity-name user1 \
     --alter --add-config 'SCRAM-SHA-256=[password=secret]'
```

3. SASL/GSSAPI (Kerberos)
   - Requires Kerberos infrastructure
   - Check: Kerberos ticket validity

```
   klist  # List current tickets
   kinit -R  # Renew ticket
```

Diagnosis:

```
# Enable JAAS logging
export KAFKA_DEBUG_OPTS="-Dcom.sun.security.sasl.debug=true"

# Check broker's JAAS configuration
grep "KafkaServer" /path/to/jaas.conf
```

5.3 Authorization (ACL) Issues

Problem: Access denied despite correct authentication

Error: TOPIC_AUTHORIZATION_FAILED

Diagnosis & Fix:

```
# List current ACLs
kafka-acls.sh --bootstrap-server localhost:9092 --list

# Grant permissions
kafka-acls.sh --bootstrap-server localhost:9092 \
  --add --allow-principal User:alice \
  --operation Read,Write \
  --topic my-topic

# Grant consumer group permissions
kafka-acls.sh --bootstrap-server localhost:9092 \
  --add --allow-principal User:alice \
  --operation Read \
  --group my-consumer-group

# Grant cluster admin
kafka-acls.sh --bootstrap-server localhost:9092 \
  --add --allow-principal User:alice \
  --operation ClusterAction \
  --cluster
```

6. Monitoring & Diagnostics

6.1 JMX Metrics Overview

Key Broker Metrics:

Metric	Purpose	Alert Threshold
`BytesInPerSec` / `BytesOutPerSec`	Throughput	Monitor trends
`FetchConsumerTotalTimeMs`	Consumer fetch latency	> 500ms sustained
`ProduceLocalTimeMsMean`	Producer latency	> 100ms sustained
`UnderReplicatedPartitions`	Data risk	> 0 (critical)
`OfflinePartitionsCount`	Unavailable partitions	> 0 (critical)
`ISRShrinks` / `ISRExpands`	Replica issues	Track frequency
`NetworkProcessorAvgIdlePercent`	Network capacity	< 20% = near limit
`RequestHandlerAvgIdlePercent`	I/O capacity	< 20% = near limit

Key Consumer Metrics:

Metric	Purpose	Alert Threshold
`ConsumeRate`	Messages/sec consumed	Monitor trends
`FetchLatencyAvg`	Time to fetch batch	> 100ms
`CommitLatencyAvg`	Offset commit time	> 50ms
`JoinRate`	Consumer joins/sec	Spike = rebalancing
`SyncRate`	Group syncs/sec	Spike = rebalancing

6.2 Diagnostic Tools

1. Confluent Diagnostics Bundle Tool

```
# Generate comprehensive diagnostic bundle
confluent-diagnostics.sh --broker-list localhost:9092 \
  --output-directory /tmp/kafka-diagnostics

# Bundle includes:
 - JMX metrics snapshots
 - Broker configurations
 - Broker and system logs
 - Zookeeper state (if applicable)
 - Topic/partition metadata
```

2. Control Center UI

- Navigate to: `http://localhost:9021`
- Sections:
  - Cluster: Overall health, broker status
  - Topics: Partition distribution, replication
  - Consumer Groups: Lag, throughput
  - Alerts: Predefined issue notifications

3. Command-Line Tools

```
# Topic metadata
kafka-topics.sh --bootstrap-server localhost:9092 --describe

# Consumer group details
kafka-consumer-groups.sh --bootstrap-server localhost:9092 \
  --group my-group --describe --members

# Replication verification
kafka-replica-verification.sh --broker-list localhost:9092 \
  --topic-white-list ".*"

# Broker API versions
kafka-broker-api-versions.sh --bootstrap-server localhost:9092
```

6.3 Log Analysis

Critical Log Patterns:

1. Broker Startup Issues

```
   [KafkaServer id=1] started successfully -> Broker ready
   Fatal error during KafkaServer startup -> Fatal error
   ERROR Error starting socket server -> Port/binding issue
```

2.Replication Problems

ISR for partition [topic-0] shrinks to [1,2] -> Replica lag

Replica 3 is not in ISR for partition [topic-0] -> Replica offline

Leader election took 5234ms -> Slow election

3. Security Issues

[SaslAuthentication] Failed authentication -> SASL failure

NOT_AUTHORIZED -> ACL denial

SSL/TLS: Unsupported or unrecognized SSL message -> SSL config error

7. Performance & Tuning Issues

7.1 Throughput Bottlenecks

Problem: Cluster not achieving expected throughput

Diagnosis Steps:

1. Identify Bottleneck Layer
   - Network: Monitor `BytesInPerSec` vs. network capacity
   - Disk I/O: Monitor `iowait` and disk latency
   - CPU: Monitor `RequestHandlerAvgIdlePercent`
   - Memory: Monitor GC frequency and pause time

2. Network Bottleneck Solutions

- Increase `num.network.threads` (default: 3)
- Enable batching: `batch.size=32768` (producer)
- Compress: `compression.type=snappy`

3. Disk I/O Bottleneck Solutions

- Increase `num.io.threads` (default: 8)
- Use faster disks (SSD)
- Tune OS: increase `vm.dirty_ratio` and `vm.dirty_background_ratio`

4. CPU Bottleneck Solutions

- Compression trade-off: CPU vs. network bandwidth
- Profile application code
- Increase broker instances (horizontal scaling)

7.2 Latency Issues

Problem: High end-to-end latency (producer → broker → consumer)

Measurement Points:

```
Producer    Broker    Consumer
 ↓          ↓         ↓
[send]→[enqueue]→[persist]→[fetch]→[consume]
```

Tuning Strategies:

Component	High Latency Cause	Solution
Producer	Batching wait	Reduce `linger.ms`
Producer	Acks wait	Change to `acks=1` (less safe)
Broker	Replication sync	Reduce `min.insync.replicas` (risky)
Consumer	Polling interval	Increase `max.poll.records`
Consumer	Processing time	Optimize application logic

7.3 Memory Management

Problem: Frequent GC pauses or OutOfMemoryError

JVM Tuning:

```
# In kafka-env.sh or server startup
export KAFKA_HEAP_OPTS="-Xmx4G -Xms4G"
export KAFKA_JVM_PERFORMANCE_OPTS="-XX:+UseG1GC \
  -XX:MaxGCPauseMillis=20 \
  -XX:InitiatingHeapOccupancyPercent=35 \
  -XX:G1HeapRegionSize=16M"
```

Broker-Level Tuning:

- `log.flush.interval.messages`: Reduce to flush pages sooner (default: disabled)
- `log.segment.bytes`: Smaller segments reduce memory pressure (default: 1GB)
- `replica.socket.receive.buffer.bytes`: Default 64KB; reduce if memory-constrained

8. Common Error Codes & Solutions

Error Code Reference

Code	Name	Cause	Solution
0	NONE	Success	N/A
1	OFFSET_OUT_OF_RANGE	Consumer seeks past latest offset	Reset to latest: `kafka-consumer-groups.sh --reset-offsets --to-latest`
2	INVALID_COMMIT_OFFSET	Invalid offset commitment	Check consumer logic; may need offset reset
3	NOT_COORDINATOR_FOR_GROUP	Client queried wrong broker	Client automatically retries with correct coordinator
5	LEADER_NOT_AVAILABLE	No partition leader	Wait for leader election; check broker health
6	NOT_LEADER_FOR_PARTITION	Stale broker in metadata	Producer/consumer auto-update metadata
7	REQUEST_TIMED_OUT	Broker slow/unresponsive	Increase timeouts; check broker resources
10	MESSAGE_TOO_LARGE	Message exceeds max size	Increase `message.max.bytes` (broker) and `max.request.size` (client)
31	POLICY_VIOLATION	Topic creation violates policy	Check `create.topic.policy.class.name`
33	INVALID_PRINCIPAL_TYPE	Unknown principal type in ACL	Use correct format: `User:alice`, `User:*`
40	SASL_AUTHENTICATION_FAILED	Wrong SASL credentials	Verify username/password or Kerberos setup
41	UNSUPPORTED_SASL_MECHANISM	Broker doesn't support mechanism	Configure matching SASL mechanism on client and broker
45	TOPIC_AUTHORIZATION_FAILED	ACL denies operation	Grant required permissions via `kafka-acls.sh`

| 47 | SSL_HANDSHAKE_FAILURE | SSL/TLS error | Verify certificates; check SSL configuration |

9. Best Practices & Preventive Measures

9.1 Operational Best Practices

1. Capacity Planning
   - Monitor and forecast disk growth (log retention)
   - Size brokers for 70-80% capacity max
   - Plan for 3x peak throughput headroom

2. Configuration Management
   - Version control all configurations
   - Document non-default settings
   - Test changes in staging before production

3. Monitoring Strategy
   - Set up alerts for critical metrics:
     - UnderReplicatedPartitions > 0
     - OfflinePartitionsCount > 0
     - ConsumerLag > threshold
     - RequestHandlerAvgIdlePercent < 20%
     - DiskUsage > 80%
   - Establish runbook for each alert

4. Backup & Disaster Recovery

   - Regularly backup ZooKeeper data (or KRaft snapshots)

   - Document recovery procedures

   - Test recovery in staging environment

9.2 Configuration Best Practices

```
properties
# Broker Configuration (server.properties)

# Replication & Durability
min.insync.replicas=2  # Ensure quorum writes
default.replication.factor=3  # Tolerate 1 broker failure
unclean.leader.election.enable=false  # Prevent data loss

# Performance
num.network.threads=8  # Adjust based on cores
num.io.threads=8  # Disk I/O workers
num.replica.fetchers=4  # Replication parallelism
compression.type=snappy  # Default compression

# Timeouts & Health
replica.lag.time.max.ms=30000  # Max replica lag
group.min.session.timeout.ms=6000  # Consumer session timeout
group.max.session.timeout.ms=300000  # Max session timeout

# Retention
log.retention.hours=168  # 7 days
log.cleanup.policy=delete  # or 'compact' for compaction
log.segment.bytes=1073741824  # 1GB segments

# Monitoring
auto.create.topics.enable=false  # Require explicit topic creation
allow.unclean.shutdown=false  # Ensure clean shutdown
```

9.3 Producer Best Practices

```properties
# Reliability
acks=all  # Wait for all replicas
retries=Integer.MAX_VALUE  # Unlimited retries
max.in.flight.requests.per.connection=1  # Ensure ordering

# Performance
batch.size=32768  # Batch size in bytes
linger.ms=100  # Wait time for batching
compression.type=snappy  # Reduce network bandwidth

# Fault Tolerance
request.timeout.ms=30000  # Request timeout
delivery.timeout.ms=120000  # Total delivery timeout
enable.idempotence=true  # Prevent duplicates
```

9.4 Consumer Best Practices

```properties
# Parallelism
max.poll.records=500  # Messages per poll
fetch.min.bytes=1024  # Min fetch size
fetch.max.wait.ms=500  # Max wait time

# Session Management
session.timeout.ms=30000  # Consumer session timeout
heartbeat.interval.ms=10000  # Heartbeat frequency
max.poll.interval.ms=300000  # Max processing time

# Offset Management
enable.auto.commit=true  # Auto-commit or manual
auto.commit.interval.ms=5000  # Commit frequency
auto.offset.reset=earliest  # or 'latest' / 'none'

# Partition Assignment
partition.assignment.strategy=org.apache.kafka.clients.consumer.CooperativeStickyAssignor
```

9.5 Troubleshooting Checklists

When Brokers Are Down:

- Check system logs: `/var/log/syslog`
- Verify disk space: `df -h`
- Check open file limits: `ulimit -n`

- Verify port availability: `netstat -tuln`
- Review broker logs: `tail -200 server.log`
- Check JVM memory: `jps -m`
- Verify network connectivity to ZK/KRaft: `nc -zv zk-host`

When Consumers Are Lagging:

- Check consumer processing time
- Verify consumer instance count vs. partitions
- Monitor broker metrics (capacity)
- Check for rebalancing frequency
- Verify network latency: `ping broker`
- Review consumer logs for exceptions
- Check topic throughput vs. consumer rate

When Replication Is Unhealthy:

- Verify all brokers are up: `kafka-broker-api-versions.sh`
- Check ISR status: `kafka-topics.sh --describe`
- Monitor replica lag: `kafka-replica-verification.sh`
- Check inter-broker connectivity: `nc -zv broker-host`
- Verify broker configuration compatibility
- Review broker logs for network errors
- Check disk space on all brokers

10. Advanced Troubleshooting Scenarios

Scenario 1: Cascading Broker Failures

Situation: Broker 1 goes down; shortly after, brokers 2 and 3 become unresponsive.

Root Cause Analysis:

- Broker 1 fails → Partitions lose leader
- Leader election triggers on brokers 2 & 3
- All brokers simultaneously busy → Network timeout from ZooKeeper
- ZooKeeper considers brokers 2 & 3 dead
- Cascading failure

Resolution:

1. Restart broker 1 first (usually enough)
2. If not, restart brokers sequentially (30-60s between each)
3. Increase ZooKeeper session timeout: `zookeeper.session.timeout.ms=40000`
4. Increase broker's ZooKeeper sync time: `zookeeper.sync.time.ms=10000`

Scenario 2: Producer Deadlock with min.insync.replicas

Situation: Producer sends message; hangs indefinitely.

Root Cause:

- `min.insync.replicas=2` configured
- Only 1 broker currently in ISR (other is slow)
- Producer waits for 2 acknowledgments; never arrives
- Timeout occurs after `request.timeout.ms`

Resolution:

1. Fix underlying broker issue (check slow broker)
2. Temporarily reduce `min.insync.replicas` if critical
3. Set `request.timeout.ms` to finite value
4. Implement client-side timeout handling

Scenario 3: Consumer Group Stuck During Rebalancing

Situation: Consumer group shows `PreparingRebalance` state for hours.

Root Cause:

- Group coordinator unavailable or unresponsive
- Consumer fails during rebalance (throws exception)
- Broker GC pause during assignment computation

Resolution:

1. Check coordinator broker: `kafka-consumer-groups.sh --describe --group my-group`
2. Verify coordinator broker is healthy
3. Restart consumer application
4. Manually trigger rebalance: `kafka-consumer-groups.sh --reset-offsets --to-latest --group my-group`

Conclusion

Troubleshooting Confluent Kafka effectively requires:

1. Understanding Architecture: Know how brokers, replicas, and consumers interact
2. Systematic Approach: Gather facts, isolate issues, apply targeted fixes

3. Monitoring: Proactive metrics and logging prevent crises

4. Documentation: Maintain runbooks and configurations

5. Testing: Validate fixes in staging before production

Key takeaways:

- Replication health is critical; monitor under-replicated partitions
- Consumer lag indicates end-to-end system health
- Configuration matters: Many issues stem from suboptimal settings
- Timeouts are often symptoms, not root causes; dig deeper
- Preventive measures (alerting, backups, capacity planning) are valuable

Keep this guide nearby for quick reference, and build experience through hands-on troubleshooting in test environments.

Additional Resources

- Confluent Documentation: https://docs.confluent.io/
- Apache Kafka Documentation: https://kafka.apache.org/documentation/
- Kafka Protocol Documentation: https://kafka.apache.org/protocol
- Community Forums: https://www.confluent.io/community/
- Confluent Support: https://support.confluent.io/