Ensuring Stability: A Technical Blueprint to Kafka Upgrade Testing

An Amrush White Paper

Purpose

This white paper aims to present a clear, structured technical framework for testing Confluent Kafka cluster upgrades in lower environments ensuring they can be safely applied in production.

Scope

- Cluster migrations (e.g., on-prem -> cloud).
- Major version upgrades (e.g., CP 7.3.x -> CP 7.6.x).
- Java runtime migrations (e.g., Java 8 -> Java 17).

Audience

- Kafka Administrators
- DevOps Engineers
- Site Reliability Engineers

Compatibility Analysis

Initial step to make sure your infrastructure aligns with the target version.

1. Identify the current version of your Kafka cluster including the versions of OS, JDK, Ansible, client libraries and any other components that might be running in your environment.
2. Review all Confluent release notes between your source and target versions to understand changes, deprecations, or feature additions - particularly those that might impact client compatibility or configurations.
3. Document all the new changes, additions and features released out by Confluent and decide accordingly if you are going to implement them.
4. Pay attention to Kafka Improvement Proposals (KIPs) that introduce behavioural or protocol shifts.
5. Make sure the current versions of the components are compatible with the target version.
6. Record compatibility gaps, required configuration adjustments, and potential migration blockers and define remediation steps.
7. Special attention to confirm if there any changes in the properties file of any Kafka components.
8. Conclude the compatibility analysis based on your findings and proceed with upgrade in a lower environment.

Note: This section provides valuable context, but the white paper focuses specifically on testing the Kafka cluster post-upgrade. For compatibility analysis and upgrade justifications, consult with stakeholders, clients, and application owners to clarify the “why” and “how” of the upgrade.

Sanity

A quick surface-level validation to confirm that modifications haven't disrupted essential operations.

- Monitoring – Validate that cluster metrics are being generated as expected and consumed by the configured monitoring system.
- System Level Metrics – Verify that system-level metrics such as CPU utilization, disk usage, and Java heap memory remain within acceptable bounds, with no concerning spikes.
- Topics/ACLs – Verify that all Kafka topics remain correctly configured following the upgrade along with all topic-related access controls—either legacy ACLs or modern RBAC bindings remain intact and consistent with your cluster's pre-upgrade configuration.
- Messaging – Verify that messages are successfully produced to and consumed from the intended topics.
- Schema – Confirm that messages conform to the expected schema and maintain compatibility across schema versions.
- Ksql – Validate that KSQLDB queries run correctly, processing input streams/tables and writing outputs to the proper topics.
- Connector – Ensure Kafka Connect source and sink connectors are functioning correctly, transferring data between Kafka and external systems.
- Mode – Verify that the cluster is still operating in the expected mode—either zookeeper or kraft as configured.
- Bug – Document any identified bugs and assess their severity and potential impact to determine if they are negligible enough to safely proceed with the upgrade.

Tip: It's unnecessary to validate all existing topics, ACLs, schemas, streams, and connectors during sanity testing. Instead, create a few representative ones and focus your testing on those.

Regression Testing

A comprehensive validation that executes core functionalities to ensure that no existing behaviour has regressed due to recent changes.

Topic

- Validate all the topic operations such as creation, deletion, altering and purging.

Messaging

1. Produce to a topic.
2. Validate if the messages are stored in the order that they are produced.

3. Consume from the topic with a group.
4. Repeat the steps 1-3 for a compact topic and validate if messages with same keys are stored in the same partitions.

Schema

1. Validate schema registration and deletion.
2. Register any schema to any topic which will be registered as the first version.
3. Produce to the topic using avro console producer with schema version 1.
4. Consume.
5. Register a compatible schema and validate if the schema is being evolved as expected with version 2 being the current version.
6. Repeat steps 3-4 with schema version 2.
7. Validate the production/consumption is working as expected with schema version 1.
8. This will confirm that the existing and newly added producers/consumers are not affected by multiple versions of schema being used in the topic.

Connector

1. Test connection with another environment running in a different CP version.
2. Validate if the connector is working as expected in different scenarios and able to recover after a failover.

KSQL

1. Create a stream1 with a topic containing any schema.
2. Create another stream2 to query the output from stream1.
3. Produce to the topic and consume from the output stream.
4. Verify if the consumed messages contain the expected columns as queried.

Fault Tolerance

1. Validate if the production/consumption is happening as expected with broker failover scenarios.
2. Create two topics and a consumer group.
3. Produce to the first topic, consume with a group and re-produce to the other topic.
4. Use two scripts to produce and consume/re-produce continuously in a loop. This simulates a live streaming scenario.
5. Now, test if the live streaming is working as expected during failovers and able to recover successfully later.

Cluster Link

1. Verify if the cluster linking is working as configured along with schema exporter.

2. Cluster Linking is tested for the purpose of DR/Migration and if you are following any different approach such as storage-based backups or offline data restoration —be sure to test that approach as thoroughly as well.

Monitoring

- Verify if the monitoring system's warning/alerting/recovery is received/recorded as expected.

Load Test

A staged evaluation performed in the production-parallel or UAT environment to assess system stability under expected and peak workloads.

1. Stage this test environment such that the partitioning and replication are as same as your production environment or higher.
2. Launch producer instances and consumers with multiple threads in parallel to generate the intended load.
3. Consider the test passed if the system remains stable during peak loads without any anomalies.

Caution: When generating load, be careful not to exceed available disk space.

Performance Testing

A structured evaluation to assess cluster performance.

1. Create a topic in your test environment, explicitly specifying the following configurations at creation:

Config	***Value***
replication factor	3
partitions	3
min.insync.replicas	2
max.message.bytes	26214400

2. Launch multiple (at least 3) producer instances in parallel to write to the topic, measuring the latency per producer; similarly, for consumer instances to record the fetch time.
3. Produce with the help of kafka-producer-perf-test tool considering the following properties:

Config	***Value***
num-records	10, 100, 500, 1000, 5000, 10000, 20000, 50000, 100000
record-size	1024, 10240, 102400, 512000, 1046576, 5242880, 10485760
throughput	-1
acks	0, 1, all
batch.size	16384, 1024, 2048, 3072, 5120, 10240, 51200, 10240, 512000, 102400, 512000

linger.ms	0, 10, 20, 100, 200, 500
buffer.memory	33554432
max.request.size	27262976
compression.type	none, snappy, lz4, gqzip, zstd

Note: The values in black to be used as default for the corresponding config while testing other configs and the values in purple to be used as different values for testing respective configs.

4. Consume with the help of kafka-consumer-perf-test tool for any chosen value of messages and fetch-size.
5. Graph the average latency value of the producers for each property and fetch time of consumers for applicable properties to assess consistency and behaviour patterns.

Conclusion

You're now well-positioned to proceed with upgrading your production environment following the successful implementation of these tests in lower environments.

Implementation context

Tested during CP 7.3.4 → 7.6.0 upgrade, with a Zookeeper-based cluster.

Contact

Email: amrush.abilash@gmail.com

LinkedIn: https://www.linkedin.com/in/amru-sh