NetApp Instaclustr

White Paper

THE ULTIMATE

Apache Kafka migration guide

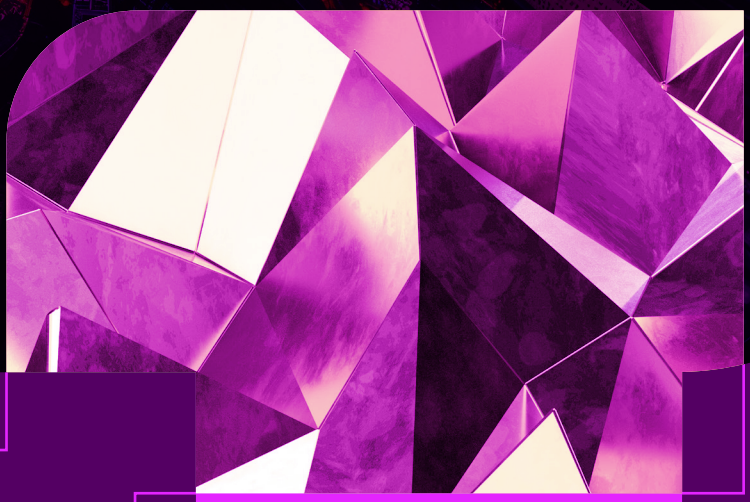


Table of contents

INTRODUCTION TO *migrating Kafka*

Our team of experts routinely assist with Kafka migrations and provide comprehensive guidance on the methodologies best suited for different environments. We have a long record of success when it comes to **migrating Apache Kafka in some of the most demanding and complex production scenarios**. This document outlines several proven approaches and explains how they can be effectively applied.

For customers of our managed platform, we work with you to determine the best migration method suitable for your needs. From initial planning through execution, we ensure the process is efficient, reliable, and aligned with your operational requirements using one of the methods detailed below.

Of course, Apache Kafka implementations are always unique, so this article is not intended to be a step-by-step migration guide. Rather, it provided an overview of different migration options and strategies available.

PLANNING *considerations*

Given the multiple approaches available for migrating Kafka clusters, the following questions are intended to help determine the most appropriate method for your specific use case. Note that switching clusters will always incur some downtime when producers/consumers restart to use the new cluster, however this can be mitigated with careful planning.

- Is any level of service interruption acceptable during migration? If so, what is the maximum allowable downtime for your applications?
- Can previously consumed messages be safely ignored or discarded post-migration?
- Will applications need to reprocess messages from the beginning of the topic to maintain functionality or state consistency?
- Is your system capable of tolerating duplicate messages without adverse effects?
- Is any data loss acceptable? If not, do you have mechanisms in place to regenerate or recover missing messages?
- Can tools like MirrorMaker or Kafka Connect be used to replicate messages from the source to the target cluster during migration?

MIGRATION *methods*

Drain-out

Set up a new cluster and move producers first, move consumers over as they complete consuming messages for their topic(s), then decommission the original cluster

Replication (via MirrorMaker 2)

Use MM2 to replicate messages between two clusters until all clients are moved over, then decommission the original cluster

Stretch Cluster

Expand the cluster to replicate data to a new one, then decommission the original cluster

Backup & Restore

Create a backup of the original cluster, then restore it to a new cluster

Migration methods comparison

Each migration method comes with its own strengths and weaknesses that may impact which is best for the target environment. Below is a summary of the key impacts of each method on cluster availability and idempotency.

	Drain Out	**Replicate via MM2**	**Stretch Cluster**	**Backup & Restore**
Incurs Downtime longer than a few hours?	No	No	No	**Yes**
Full Message history available after Migration?	No	**Yes**	**Yes**	**Yes**
Possible message duplication?	No	**Yes**	No	No

	Drain Out	Replicate via MM2	Stretch Cluster	Backup & Restore
Some messages may be lost or will need to be re-produced?	**Yes**	No	No	Yes
How much effort is required for planning and execution?	**Low**	Medium	High	Medium
How much extra infrastructure is needed?	**Medium**	High	Medium	Low
How long is the average migration going to take?	**Short**	Short	Long	Medium

Drain-out

The drain-out approach involves provisioning a new cluster to run in parallel with the existing one, with all the topics and Access Control Lists (ACLs) configured to match the source cluster, enabling applications to transition seamlessly when ready. This approach does not require copying existing data from the source cluster.

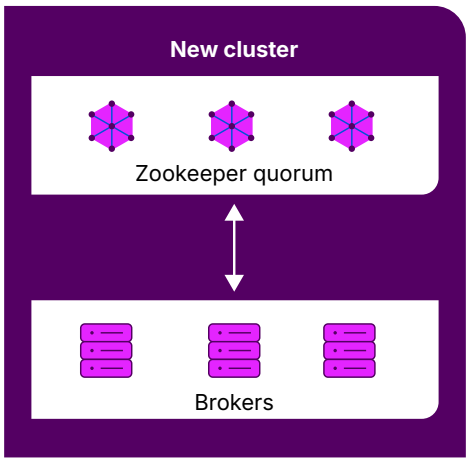
The drain-out method is considered one of the simplest migration approaches, as it primarily involves updating application configurations, specifically the bootstrap servers and security settings, once the new cluster is fully operational. By maintaining identical topic configurations (e.g., partition count, replication factor, retention policies, and message size limits), the risk of service disruption or unexpected behavior during the transition is minimized.

Pros & cons

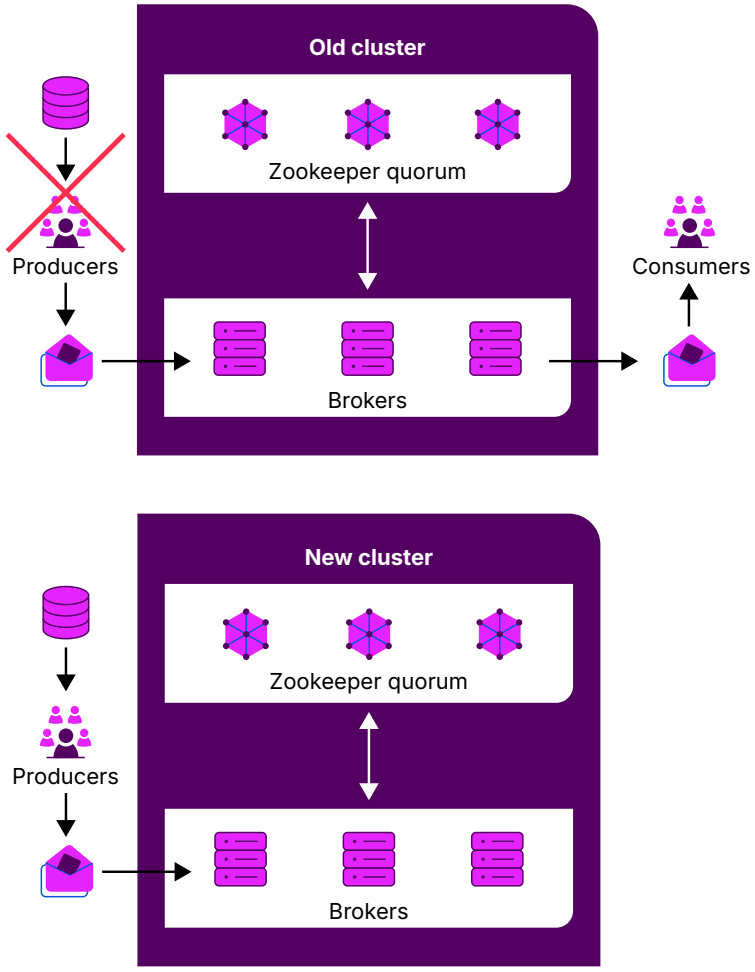
Pros	Cons
• Simple and relatively fast • No need for data consistency checks • Pre-migration data is retained on the source cluster, and can remain online until the new cluster is stable • Easy roll-back planning, by reverting to the source cluster • Doesn't incur downtime but requires careful client management to avoid data loss or service disruption.	• Topic message data from the source cluster is not available on the new cluster • Risk of data loss if consumers are re-directed to the new cluster before producers. • Migration can be time consuming and complex involving large number of clients and creating significant operational burden and directly impacts downtime dependent on how quickly the reconfiguration process can be completed.

Steps

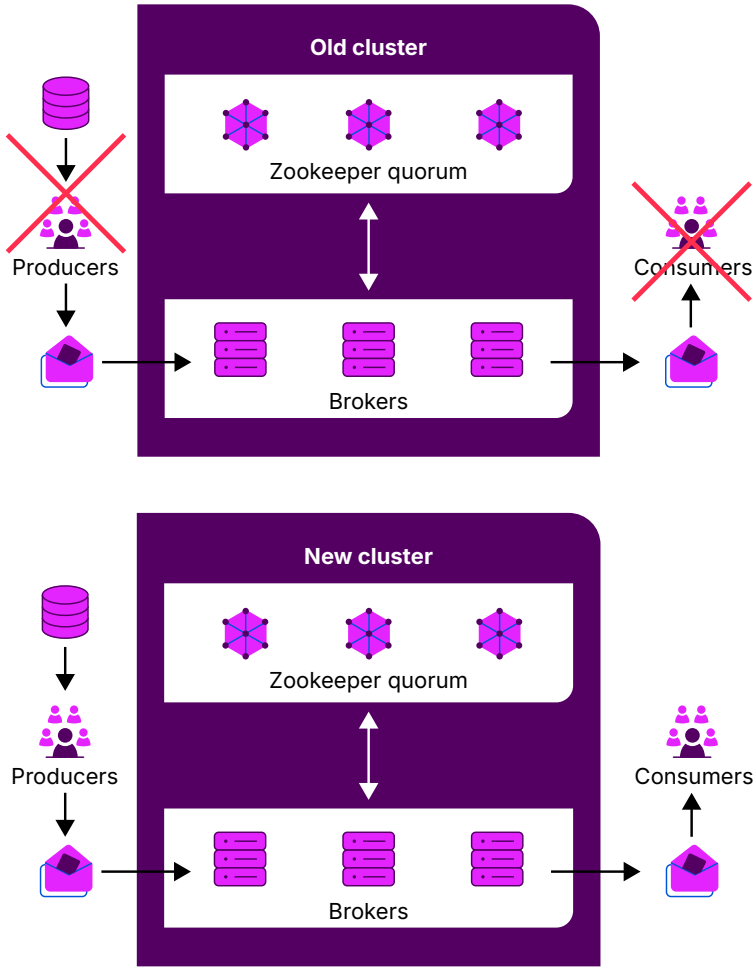
1. Deploy a new cluster to run in parallel with the existing one. Create all required topics in the cluster ensuring that topic configurations match the source cluster – Partition count, Replication Factor (RF), Retention policies and Message size limits and apply appropriate ACLs to allow applications seamlessly switch over without disruption when ready.



2. Stop producers connected to the source cluster and re-direct them to the new cluster while the consumers are still subscribed to the source cluster to continue processing existing messages.
   Monitor consumer lag on the source cluster and ensure it reaches zero before proceeding. This will allow all messages are consumed and no data is lost during transition.



3. Once consumer lag is zero, stop consumers on the source cluster and update configuration auto.offset.reset to earliest, allowing the consumers to read messages from the beginning of the offset range and then point the consumer to the new cluster's bootstrap servers.



Alternative Drain-Out: Leaving the producers stopped

If temporary downtime for producers is acceptable, an alternative approach is to keep producers offline until the consumers have fully drained the existing cluster.

- Provision a new Kafka cluster to run alongside the existing one. Create all required topics in the cluster ensuring that topic configurations match the source cluster – Partition count, Replication Factor (RF), Retention policies and Message size limits and apply appropriate ACLs to allow applications seamlessly switch over without disruption when ready.
  Create all the required topics and set ACLs in the new cluster to match the source cluster, which will make it possible for applications to switch over when you are ready.
- Stop producers connected to the source cluster while the consumers are still subscribed to the source cluster to continue processing existing messages. Monitor consumer lag on the source cluster and ensure it reaches zero before proceeding. This will allow all messages to be consumed and ensure no data is lost during transition.

  Note: While producers remain offline, no new messages will be ingested into either the source or the new cluster.
- Once consumer lag is zero, stop consumers from source cluster and update configuration auto.offset.reset to earliest, allowing the consumers to read messages from the beginning of the offset range and then point the consumer to the new cluster's bootstrap servers.
- Update producer configurations to connect to the new cluster and restart producers to resume message publishing into the new cluster.

Replication via MirrorMaker 2

In this approach, a new Kafka cluster is provisioned and data from the source cluster is replicated to it using MirrorMaker 2. After the initial synchronization, it continues to replicate data in near real time, ensuring both clusters remain aligned during the migration window. Once all the client applications have been fully migrated to the new cluster, the source cluster can be safely decommissioned.

MirrorMaker 2 (MM2) is a widely adopted & effective way for zero-downtime Kafka migrations, particularly suited for complex or large scale environments. However, it introduces additional operational complexity compared to simpler strategies like the drain-out method. For deeper insights, refer to detailed documentation or blog posts on MirrorMaker 2 architecture and deployment best practices Apache Kafka MirrorMaker2 .

Beyond the setup and configuration of MirrorMaker 2, client applications may require code changes, ranging from minor to significant, depending on their architecture and how they interact with Kafka.

- **Minor changes** typically involve adjustments to topic naming conventions. In MirrorMaker 2, mirrored topics often include a prefix (e.g., source-cluster.topic-name), which clients must account for when subscribing or publishing to topics in the new cluster. However, to simplify migration and reduce client-side changes, it's also possible to configure it to retain original topic names in the target cluster, eliminating the need for prefix handling.
- **Major changes** may arise due to **offset discrepancies**. Offsets in the source cluster do not directly map to those in the target cluster. While MirrorMaker 2 supports offset translation for Kafka-managed offsets, this can become complex if your applications store offsets externally (e.g. in a database or custom store). Integrating MirrorMaker 2's offset translation logic in such cases may require substantial development effort to ensure consistency and prevent data duplication or loss. Additionally, With the introduction of KIP-656, MirrorMaker 2 now supports **exactly-once delivery semantics**, enhancing reliability and reducing the risk of message duplication during replication – making it a more robust choice for critical data pipelines.

Pros & cons

Pros	Cons
• Keep the source and target cluster in sync during the migration • Supports to produce/consume from its last known state in the source cluster • Translates and synchronizes consumer group offsets to the target cluster • Facilitates smooth client switchover without data loss • Automatically replicates the __ schemas topic, avoiding schema republishing • Enables zero-downtime migration for most use cases	• Requires installation and configuration of MirrorMaker 2 • Offset translation can be complex if offsets are stored externally • Additional monitoring and operational overhead and associated infrastructure costs may be required.

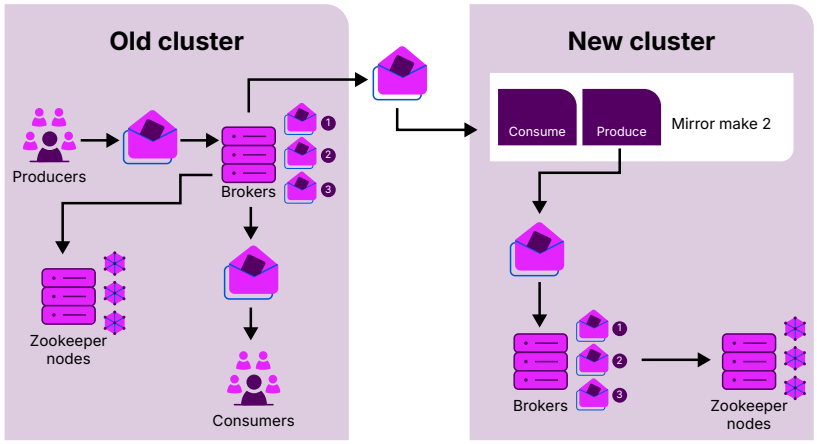
Special considerations

If offset sync is not configured (sync.group.offsets.enabled is set to false), applications may consume duplicate messages.

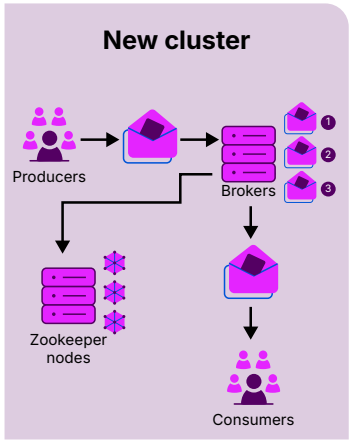
Steps

1. Set up MirrorMaker 2 on the new target cluster and configure it to replicate data from source to target cluster.

2. Ensure replication includes both topic data and consumer group offsets (by setting sync.group.offsets.enabled=true if offset sync is required).



3. Re-configure producers and consumers to point to the target cluster's bootstrap servers.



4. Once all the clients are successfully switched to the new cluster, disable MirrorMaker 2 and decommission the source cluster.

Stretch cluster

This approach involves extending Kafka brokers across multiple regions or data centers to facilitate a seamless migration which enables clients to continue interacting with both source and target cluster until the final cutover.

The migration process begins by adding a target cluster (including Zookeepers and Brokers) into the existing cluster setup. Using partition reassignment, topics are then gradually shifted to the target nodes. Once all clients are exclusively connected to the target nodes, the source cluster can be safely decommissioned.

Pros & cons

Pros	Cons
• Ensures data consistency and integrity throughout the migration • Enables near zero-downtime transitions • Very reliable and highly suitable for large-scale Kafka deployments • Offsets remain consistent across the stretch, eliminating the need for offset translation and simplifying consumer behavior.	• Requires significant effort to align the target nodes with existing cluster settings including security, ACL's and replication policies. • Cross-region replication introduces latency and requires a highly reliable, low-latency network to avoid ISR and controller issues.

Special considerations

Although the cluster downtime is avoidable, there is a brief transition period during Zookeeper handover when original Zookeepers go offline before the new ones take over where metadata requests may fail, leading to client retries. Planning for this transition is crucial to minimize impact.

The new Zookeeper nodes will be added as Observers to reduce the time required to transition to the new ensemble, however this method requires tight timing to avoid an outage during the transition itself.

Steps

Example Cluster IDs

Old Cluster (A)		New Cluster (B)	
Broker IDs	**Zookeeper IDs**	**Broker IDs**	**Zookeeper IDs**
1	1	4	4
2	2	5	5
3	3	6	6

1. Add new zookeeper nodes (4,5 and 6) to the existing quorum as observers by updating the zoo.cfg file
2. Add new brokers (ID's 4, 5 and 6) to the existing cluster with zookeeper.connect directed at the new zookeepers
3. Use Kafka's partition reassignment tool to move topic partitions to the new brokers (4,5 and 6).
4. Reconfigure clients to point to the new cluster's bootstrap servers.
5. Shutdown the old brokers (IDs 1, 2, and 3).
6. Shutdown all Zookeepers
7. Update zoo.cfg to set the new Zookeepers as participant and remove the old Zookeeper references
8. Start the new Zookeepers and watch the logs carefully for errors

Alternative Stretch Cluster: Participant Zookeepers

While it can introduce added latency during the process, the Zookeeper ensemble can be stretched across to the new cluster in addition to the brokers. The process is very similar, but does require additional restarts for updating the brokers configuration.

1. Add new brokers (ID's 4, 5 and 6) to the existing cluster, increasing the total broker count to six.
2. Use Kafka's partition reassignment tool to move topic partitions to the new brokers (4,5 and 6).
3. Reconfigure clients to point to the new cluster's bootstrap servers.
4. Shutdown the old brokers (IDs 1, 2, and 3).
5. Add new zookeeper nodes (4,5 and 6) to the existing quorum by updating the zoo.cfg file.
6. With six zookeeper nodes, the quorum can now tolerate up to two node failures.
7. Update the zookeeper.connect property in each broker's server. properties to reference the new zookeeper ensemble, followed by a rolling restart of all brokers.
8. Removing a non-leader node e.g. 1 from the ensemble. Update zoo. cfg on nodes 2,4,5 and 6 to exclude node 1 and restart each node after the change. (Important: Always remove the leader node last to avoid quorum instability.
9. Apply the same process to remove nodes 2 and 3, ensuring the leader is handled last in each case.

Backup and restore

This approach involves temporarily stopping the source cluster to create a complete backup and then restoring it on a new set of nodes with minimal downtime.

To reduce service interruption, the process begins with the initial data sync (pre-sync) while the source cluster remains active. Although this early sync may not capture all data, it significantly reduces the volume needed during the final sync.

Kafka data directories can be archived using compression formats like tar or zip and stored in cloud-accessible locations such as Amazon S3. Once the initial backup (pre-sync) is nearly complete, stop the source cluster to finalize the data sync. This approach shortens the downtime window by limiting the data transfer required after the cluster is stopped.

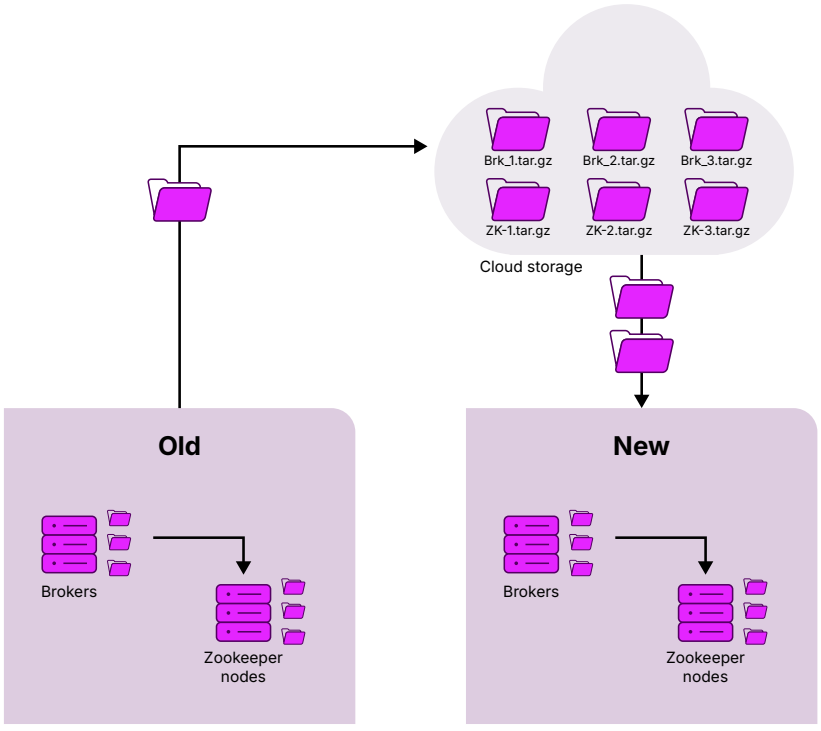
After the data is copied to the new cluster, the new cluster is started, then producers and consumers are reconfigured to connect to the new cluster completing the migration.

Pros & Cons

Pros	Cons
• Simple to implement without requiring complex procedures • Ensures data consistency with no risk of message duplication • Preserves offsets and clients can resume consumption with minimal changes post-migration.	• **Requires full downtime** during the backup and restore process impacting availability. • Operationally intensive for large clusters or high-throughput topics.

Steps

1. Set up the new cluster but keep them offline initially.
2. Update necessary configurations on new nodes as required such as hostnames and IP addresses.
3. Create a backup destination accessible by all nodes (e.g., Amazon S3).
4. Perform an initial backup(pre-sync) of the data while the source cluster is still running.
5. Restore the backup to the new nodes.
6. Temporarily stop all producers and consumers to prevent further data changes.
7. Stop the existing cluster to finalize the migration.
8. Capture any remaining data changes from the source cluster (differential backup).
9. Restore the differential backup to the new nodes.
10. Start the new cluster with restored data.
11. Reconnect producers and consumers to the new cluster.



CONCLUSION

Kafka migrations are unique, and no single approach is universally optimal. Each migration scenario presents unique requirements, making thorough planning before implementation the most critical factor for success. In this blog, we have covered four key approaches - Drain-out, MirrorMaker 2, Stretch Cluster, and Backup & Restore, highlighting their respective trade-offs in downtime, complexity, and operational overhead.

In our managed service, we frequently recommend the Stretch Cluster method for its balanced approach to complexity and uptime. For environments where zero downtime is non-negotiable, MirrorMaker 2 provides a strong alternative, though it requires additional infrastructure and operational effort. Ultimately, selecting the right strategy depends on your business priorities and technical constraints.

With NetApp Instaclustr, these migration options are part of the onboarding process to ensure a seamless and reliable transition. If you are planning a Kafka migration or require expert guidance, contact us to get started, and explore more on our Kafka Product page.

NetApp® Instaclustr specializes in open source technologies for enterprises. Our managed platform streamlines data infrastructure management, backed by experts who ensure ongoing performance, scalability, and optimization. This enables companies to focus on building cutting edge applications at lower costs.