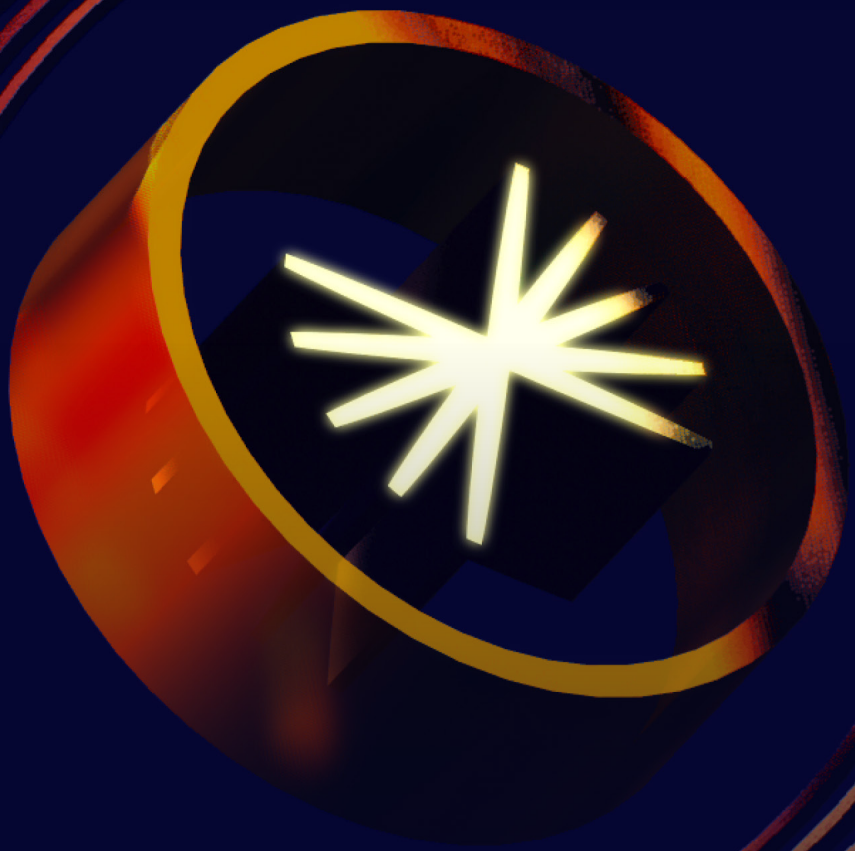
CONFLUENT



MIGRATION KIT

Migrating from Kafka services to Confluent

Migrating from Open Source Kafka to Confluent Cloud Whitepaper 4

Your in depth guide for migrating to Confluent Cloud

Summary 05
Introduction 06
Why Apache Kafka vs. Confluent Cloud 07
Migration Overview 08
Plan 09
Pre-migration Checklist 09
Define Migration Timeline 09
Define Migration Scope 10
Define Acceptable Downtime 14
Define Processing Requirements 15
Review Potential Security Changes 15
Select Network Connectivity 17
Select Confluent Cloud Cluster Type 18
Select Optimal Data Replication Tool 18
Confluent Cluster Linking 19
Confluent Replicator 22
MirrorMaker 2.0 24
Select Optimal Schema Replication Tool 25
Set Up 27
Prepare the Cluster 27
Configure and Deploy the Schema Replication Tool 28
Prepare the Clients 29
Benchmark 29
Migrate 31
Client Migration Considerations 31
Migrate without Replicating Topic Data 32
Migrate with Replicating Topic Data 34
Migrating Schema Registry 37
Validate 37
Ready to get started? 38
Appendix 39

Confluent Migration Accelerator 40

Jumpstart your data streaming journey by finding the right partner for your migration needs through Confluent's certified network of system integrators

Unpacking the Cloud Migration Process with Improving 42

Learn about Improving's Migration Workshop methodology, which covers everything from use case identification, roadmap development, and important considerations for building event-driven architectures

BigCommerce's Migration from Kafka to Confluent 43

See how BigCommerce took their business to the next level by migrating 1.6 Billion events a day to Confluent

Cerved Case Study 44

Learn how Cerved partnered with Confluent to provide instant access to business critical data to empower strategic decisions and support

Introduction

Apache Kafka® is a widely adopted open-source data streaming technology, but as organizations expand their Kafka footprint, challenges like scalability, performance, and security emerge. Confluent offers a cloud-native, complete data streaming platform that addresses these challenges, enabling organizations to accelerate innovation by freeing technical teams from low-level infrastructure.

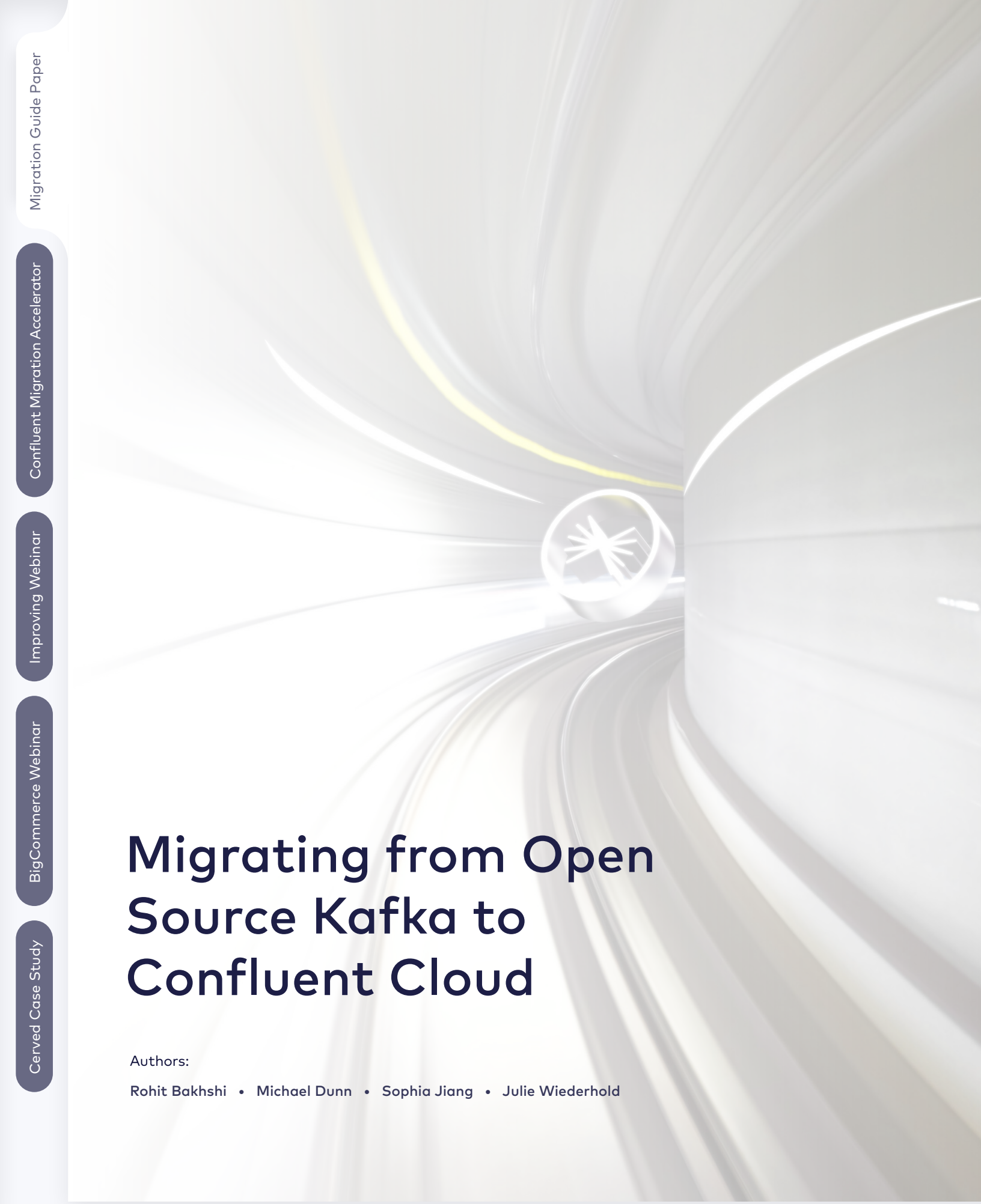
One of the most common questions we hear from organizations is, **"How do we migrate from our open-source Apache Kafka deployment to Confluent without any downtime?"** Truthfully, the answer is, it really depends on your specific environment.

This can make migrations seem daunting, but they don't have to be. This kit aims to instill confidence in those contemplating adopting Confluent. Think of this as your comprehensive guide that demonstrates how migrations are achievable (and not so scary).

While the content will focus primarily on migrations from open-source Kafka, most of the concepts can be applied when migrating from another messaging platform or a hosted or partially managed Kafka service. In this kit, you'll find:

- Best practices migration handbook that provides a "how to" framework for migrating from an existing Apache Kafka deployment
- Information about certified partner programs for additional Kafka expertise
- Real-life customer success stories that detail their personal migration journeys

Ready to get started?



Migrating from Open Source Kafka to Confluent Cloud

Authors:

Rohit Bakhshi • Michael Dunn • Sophia Jiang • Julie Wiederhold

Summary of Approach to Migrating to Confluent Cloud

A migration can be broken down into a few simple phases:

Plan: Define migration scope, objectives, and associated timelines. Build a runbook with step-by-step procedures required to migrate the cluster and clients.

- Consider business requirements, technical roadmap, client team priorities and ability to execute as factors that will impact your timeline.
- Assess your existing cluster(s) by reviewing cluster configuration, topic topology and data model, and Kafka client configuration and performance.
- Determine the data replication requirements (i.e. clients ability to process duplicates, clients ability to tolerate missed messages).
- Consider tackling technical debt from the existing deployment, such as optimizing and improving the clients, workload, data model, data quality, and cluster topology.
- Determine the amount of downtime that is acceptable for clients, executing production workload migration during a period of low traffic to mitigate business interruption.
- Based on the data replication requirements, determine the best replication tool and migration pattern. Cluster Linking is the recommended data replication tool for most use cases.
- Review security changes and implement additional security controls as needed (e.g., encryption, authorization and authentication).
- Select the Confluent Cloud cluster type based on workload and networking requirements.

Set up: Provision and configure the Confluent Cloud cluster(s).

- Iteratively set up clusters to more easily adjust configurations, improve ROI, and reduce migration costs.
- Leverage the Confluent Terraform provider to deploy and manage Confluent Cloud infrastructure so you can build, change, and version your cloud data infrastructure in a safe and efficient way.
- If required, configure and deploy the appropriate data and schema replication tools.
- All clients will need to update their bootstrap server and security configurations to connect to the Confluent Cloud cluster. It is recommended to have central repositories for configurations and secrets.
- Conduct performance benchmarks, including any additional security configurations.

Migrate: Begin your migration with lower environments and iterate through each. Production should always be the last to migrate.

- Decide whether you want to migrate an entire cluster and associated workloads all-at-once (recommended) or in a phased approach. All-at-once avoids untangling complex client interdependencies and reduces the overall migration time.
- A migration with no data replication is the simplest. There are three common patterns: Restart All At Once, Stop-Wait-Restart, and Blue/Green.
- If data replication is required, common migration patterns are: Stop-Wait-Restart and Stop-Restart-Repeat.

Validate: Confirm your workloads are functioning as expected and that performance benchmarks are met.

Please note this is a high-level summary of the steps needed to migrate to Confluent Cloud. More details about each step can be found in the whitepaper itself.

Executive Summary

Apache Kafka® is a well-established open-source data streaming technology that has been adopted by over 80% of Fortune 100 companies. However as organizations grow their Kafka footprint, the typical challenges of open-source technology emerge. These challenges include scalability, performance, and security that limit organizations from innovating quickly, and subsequently increase their risk of a system outage.

Confluent addresses these challenges by offering a cloud-native, complete data streaming platform that is available everywhere you need it to be. Thousands of global organizations have migrated to Confluent Cloud for their Apache Kafka deployments to accelerate and de-risk their path to embracing data streaming for mission-critical workloads.

While there are many paths to running workloads with Confluent Cloud, most users start their journey toward Confluent Cloud from existing Apache Kafka deployments, deployed either on-premise or in a public cloud. In this paper, we will guide you through a framework to achieve a smooth and seamless transition from Apache Kafka to Confluent Cloud. Many of these concepts can also be applied when migrating processes from another source, such as a different messaging platform or a hosted/partially managed Kafka service. However, the scope of this paper will focus exclusively on migrating from Apache Kafka to Confluent Cloud.

Why Apache Kafka vs. Confluent Cloud

Before diving into how to migrate, it is important to understand the difference between Apache Kafka and Confluent Cloud. While Kafka is a powerful distributed system, many enterprises do not want to be in the business of managing the open-source distribution in-house.

To offer a cloud-native Kafka service means it is not enough to just put Kafka on servers in the cloud. Like any system, the design is defined by the many constraints of the deployment. The challenges of a cloud data system are quite different from a self-managed implementation of the open source project.

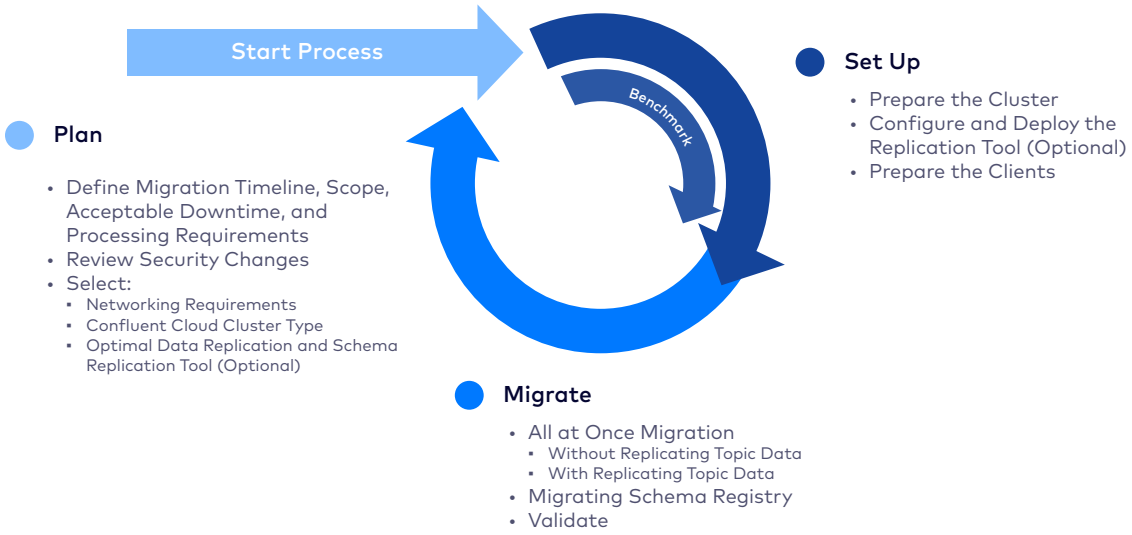
When you migrate to Confluent Cloud from an existing Apache Kafka deployment, you migrate from an open source Kafka engine to Confluent Cloud's Kora engine. Kora is fully compatible with all currently supported versions of the Kafka protocol and is designed from the ground up to be a true cloud-native, fully managed service. The existing Apache Kafka cluster is almost always designed, configured, and deployed differently from the Kora-powered Confluent Cloud cluster you are migrating to. Therefore, you should be prepared to make small adjustments to your Kafka applications to meet business requirements.

Confluent has the experience of managing thousands of customer deployments and uses that experience to optimize many system configurations that are not exposed to the user. Due to this, Confluent abstracts away much of the cluster configuration and management. Activities like software upgrades, choosing the appropriate instance type, tuning memory and CPU, and many other common tasks involved in self-managing Kafka are no longer required when using Confluent Cloud. By obfuscating complexity, Confluent provides the best in class data streaming platform with the highest SLA in the industry that is designed to meet performance requirements of most workloads. Confluent also provides a comprehensive set of enterprise-grade capabilities, reducing the need for costly development cycles and operational burdens, saving up to 60% in total cost of ownership.

Migration Overview

As with any migration, preparation is key! It is crucial to establish a clear understanding of the migration's scope and objectives, enabling your team to stay on schedule and avoid any unwanted surprises. In larger organizations, various stakeholders—such as those in security, networking, and data platforms—are typically involved. Therefore, it is advised to define key milestones and what success looks like for each stakeholder.

A migration from Apache Kafka to Confluent Cloud can be broken down into a few simple phases:



- **Plan:** Define migration scope and objectives to ensure alignment between cross-functional teams and lay a strong foundation for the migration process
    - **Runbook:** The planning stage should also include building a runbook with the step-by-step procedures required to migrate the cluster and applications
- **Set up**: Provision and configure the Confluent Cloud cluster(s)
    - **Benchmark**: Conduct performance benchmarks, including security and load testing, to ensure the new system will meet business requirements
- **Migrate:** Begin your migration with lower environments and iterate through each to move up the lifecycle chain. Production should always be the last to migrate after thorough testing and benchmarking of lower-level stages. Repeat the below steps for each cluster:
    - **Replicate Data (optional)**
    - **Migrate Clients**
- **Validate:** Confirm your workloads are functioning as expected and that performance benchmarks are met

The following sections dive deep into the important considerations required at every step of the migration process.

Plan

As the saying goes, *"If you fail to plan, you plan to fail,"* which is true of most platform migration efforts—and this is no different. To help you and your team better prepare, the following sections delve into the essential considerations needed to build a comprehensive migration plan. The most successful migrations are those that have a thought out and detailed plan.

Pre-migration Checklist

- ❏ Define the migration timeline
- ❏ Define the migration scope
  - ❏ Assess your existing clusters to understand source and target cluster topology
  - ❏ Determine topic and data replication requirements and strategy
  - ❏ Assess client serializers and deserializers that are currently in-use
  - ❏ Evaluate workload optimizations
- ❏ Define acceptable downtime
- ❏ Define processing requirements
- ❏ Review potential security changes
- ❏ Select the network connectivity
  - ❏ For the Confluent Cloud cluster
  - ❏ For connectivity between the existing Apache Kafka and Confluent Cloud Cluster
- ❏ Select the best Confluent Cloud cluster type for your workloads
- ❏ Select the optimal data replication tool (Optional)
- ❏ Select the optimal schema replication tool (Optional)
- ❏ Review the migration plan with all stakeholders to ensure alignment

Define Migration Timeline

The speed at which you move to Confluent Cloud is entirely your decision as many factors can impact the timeline, such as business requirements, technical roadmap, client team priorities and ability to execute, and others. Consider these factors to determine the intended date to complete the migration. Most migrations involve migrating an entire cluster and all associated workloads at once, or over a short window.

Each item in the Pre-Migration Checklist can impact the timeline. Be prepared to adjust your timeline as decisions or requirements are defined for each section.

Define Migration Scope

Assess Your Existing Cluster

Understanding the existing Kafka cluster deployment is critical to ensuring a smooth transition to the new Confluent Cloud cluster. This assessment should include a review of cluster configurations, the topic topology and data model, and Kafka client configurations and performance. As part of the assessment, it is prudent to identify a list of benchmarks you can use to validate the workload(s) once migrated to Confluent Cloud.

Be sure to consider the latency requirements of your existing workloads and any configuration changes you have made to achieve them. Furthermore, understand how latency is measured so that you can effectively assess it during benchmarking exercises against Confluent Cloud. Dependency planning (knowing what applications connect to your existing Kafka clusters and how they rely on each other) should also be considered, so you can be sure to target all applications that may require changes.

Recommended Inventory to Assess for the Existing Kafka Deployment:

- ❏ List of existing Kafka clusters targeted for the migration
- ❏ For every cluster:
  - ❏ List topics
    - CLI command: `kafka-topics --bootstrap-server <cluster's bootstrap> --command-config <security config> --list`
  - ❏ Total number of partitions on the cluster
    - Sum the PartitionCount across all the topics listed by the following CLI command: `kafka-topics --bootstrap-server <cluster's bootstrap> --command-config <security config> --describe`
    - Or use the JMX metric: `kafka.controller:type=KafkaController,name=GlobalPartitionCount`
- ❏ Kafka client inventory (producers/consumers)
  - ❏ List client libraries and the owning teams: Java, Go, Python, etc.
  - ❏ List data processing requirements (i.e. tolerance for duplicate messages, missed messages, etc.)
  - ❏ List performance requirements (i.e. latency, maximum consumer lag, etc.)
  - ❏ List the client dependencies[1] (i.e. topics used by which clients)
- ❏ Network configurations used for existing Kafka clusters (i.e. load balancers and listeners)
  - ❏ List source clusters that may be accessed from the public internet during the migration window[2]
- ❏ Security configurations used for existing Kafka clusters (i.e. authentication, authorization, and encryption)

[1]This information is only required when the chosen migration path is to migrate sets of clients in phases, rather than all the clients at once.
[2]This is optional and only recommended for clusters that are not already publicly accessible. Public endpoints can typically simplify the overall migration process, and some groups will create a new listener with a public endpoint if one does not yet exist.

- ❑ Establish baseline client metrics
  - ❑ Ingress and egress
    - Ingress JMX metric (per topic): `kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec,topic={topicName}`
    - Egress JMX metric (per topic): `kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec,topic={topicName}`
  - ❑ Peak active client connections
    - JMX metric (per listener): `kafka.server:type=socket-server-metrics,listener={listener_name},networkProcessor={#},name=connection-count`
  - ❑ Peak client connection attempts per second
    - JMX metric (per listener): `kafka.server:type=socket-server-metrics,listener={listener_name},networkProcessor={#},name=connection-creation-rate`
  - ❑ Peak client requests per second
    - JMX metric: `kafka.network:type=RequestMetrics,name=RequestsPerSec,request={Produce|FetchConsumer|FetchFollower}`
  - ❑ Any specific admin and governance configurations that are important in the deployment

Determine Data Replication Requirements

An important decision when migrating to Confluent Cloud is whether or not you need to migrate the data in your existing Kafka cluster to Confluent Cloud. The decision does not need to be unilateral – some of your topics may require their data to be migrated and others may not. See Select Optimal Data Replication Tool (Optional) for more information. Below is a table with example scenarios.

When to migrate data...	When to not migrate data...
✓ Data is needed for archival or replay in the destination cluster	✗ Ephemeral data used for testing

Evaluate Workload Optimization

A migration is a great time to tackle some of the technical debt that may have built up in your existing deployment, such as optimizing and improving the clients, workload, data model, data quality, and cluster topology. Each option will vary in complexity.

The table below captures some of the optimizations and improvements you can make when migrating, and how complex each of these is.

Complexity	Optimization	Benefits
Simple	**Remove unused topics (Highly recommended!)**	Over time, unused topics build up. By only migrating topics in use, you can lower the number of partitions and optimize the size of your Confluent Cloud implementation. Identifying unused topics can be done in a few different ways: 1. Parse client code 2. Finding topics with 0 throughput over a set period of time (can use the ingress/egress JMX metrics listed in the table above) 3. Finding topics with 0 data storage using the Kafka CLI: `kafka-log-dirs --describe --bootstrap-server localhost:9092 --topic-list {topic1,topic2…topicx}` 4. Using producer and consumer client metrics per topic
Simple - Moderate	**Enforce data quality with Schema Registry (Highly recommended!)**	Confluent Cloud makes it easy to enforce data quality in your topics. Our data governance suite makes it easier to search, share, and view data through our Data Portal, Stream Lineage, and Schema Registry. Applying a schema to a topic will ensure that only data meeting the schema is produced to the topic. If needed, you can write bad data to a Dead Letter Queue (DLQ) or send a custom error message using Data Contracts. If you plan to replicate data from the original Kafka cluster and also begin to leverage schemas in Confluent Cloud, the data replication solution will need to deserialize with the old libraries and serialize with new schema aware libraries. To achieve this: 1. Register schemas for each topic value and/or key in one of three supported formats: Avro, Protobuf, JSON 2. Set up the data replication tool to integrate with Confluent Cloud Schema Registry 3. Configure the tool's consumer to deserialize the messages on the origin cluster with the non-schema aware deserializer and the tool's producer to serialize the messages with a schema-aware serializer before writing to Confluent Cloud

Moderate	**Harden client security posture**	Authentication and authorization are enabled by default in Confluent Cloud. This requires API key pairs or OAuth principals to be configured for authentication, and role bindings or ACLs to be used for client authorization. A migration provides an opportunity to better isolate different clients so each is only able to perform the operations that it needs to. This involves creating separate service accounts for separate clients, configuring unique credentials, and associating specific levels of authorization to each embracing the concept of least privileges.
Moderate	**Repartition topics**	A common challenge is choosing the correct number of partitions for a topic. In many cases, one of the following occurs: 1. The number of partitions was chosen with little planning 2. The topic's actual throughput differs from the throughput used to determine the number of partitions needed to meet performance requirements In either case, repartitioning could benefit the deployment by removing unnecessary partitions, or improving client performance. The easiest solution is to repartition as part of the migration process, where new topics are created with the optimal number of partitions in Confluent Cloud prior to replicating data and/or repointing clients.
Moderate	**Upgrade client versions (Highly recommended!)**	Staying up to date with the latest Kafka client version is critical as it includes the latest bug fixes and enhancements. For example, the number of connections and requests from the clients to the cluster is a factor in Kafka resource utilization. Old versions of Kafka clients often have connection or request leaks that inflate the connections and requests to the cluster. Upgrading to the latest version will remedy these issues, as well as optimize cluster size and improve workload performance.
Moderate - Hard	**Consolidating Clusters**	It is common for an unnecessary number of Kafka clusters to be deployed within an organization. This can occur because overprovisioning is required to support the open source deployment, or due to the challenges of supporting multiple teams on a single cluster. Cluster provisioning and multi-tenancy – via role-based access-control, client quotas, etc. – are made simple in Confluent Cloud, and therefore it is recommended to consider consolidating multiple separate Apache Kafka clusters into fewer clusters in Confluent Cloud.
Moderate - Hard	**Migrate to a supported client library**	There are many available Kafka client libraries, most of which are not directly maintained by Confluent. To ensure predictable client behavior and the best support from Confluent in the event of an issue, it is recommended to use the Confluent-supported client libraries.

Define Acceptable Downtime

When migrating, it is important to determine the amount of downtime that is acceptable for the clients. For any Kafka migration, clients must be restarted to use the new bootstrap URL and security configurations specific to the destination cluster. Therefore, you must either plan for the downtime incurred when clients restart or run a blue/green deployment to achieve zero downtime. Depending on the client architecture and migration strategy, downtime can be very brief but is almost always greater than 0 when the same set of clients are repointed to the Confluent cluster. Therefore, it is recommended to execute the migration for production workloads during a maintenance window or a period of low traffic, like at night or on a weekend.

Client downtime can be categorized into two types: downtime for writes and downtime for reads. Typically, downtime for reads is more tolerable than downtime for writes since there are ways to have a consumer restart from where it left off after migrating to the Confluent cluster and quickly catch back up.

The amount of time the clients are down will be a combination of the following:

1. The speed at which you can update the client bootstrap and security configurations. The Prepare the Clients section details options for automating this to expedite the process.
2. The speed at which the client itself starts up. Clients have different start-up times depending on the implementation. Additionally, if there are specific client dependencies that dictate the order at which clients can start, there may be additional time taken to wait for predecessors to start up before a certain client can begin processing.
3. The speed at which any other migration-specific steps can be executed that are prerequisites to the clients being restarted against the Confluent cluster.

Zero Client Downtime

If downtime cannot be tolerated for any of the clients, then a blue/green deployment based migration can be used. A blue/green deployment is a client release model that gradually transfers user traffic from a previous version of a client or microservice to a nearly identical new release—both of which are running in production. In the case of a Kafka cluster migration, parallel sets of clients would be run against both the original cluster and the Confluent Confluent Cloud cluster.

Define Processing Requirements

Data processing requirements for each client must be well defined. It is critical to know if clients can effectively process duplicates or tolerate missed messages so that you can select the appropriate data replication tool and client migration pattern.

	Data is migrated to the Confluent cluster	**Data is not migrated to the Confluent cluster**
Can tolerate duplicates but not miss messages	Start the producer against the Confluent cluster before the consumer. The consumer can be started from: • The earliest offset • An offset aligned to a timestamp before the consumer was stopped • A translated offset	Start the producer against the Confluent cluster before the consumer. The consumer should be allowed to process all messages on the original cluster (i.e. consumer lag=0) and then can be started from the earliest offset on the Confluent cluster. There should be 0 to a few duplicates in this scenario unless the producer is not stopped gracefully.
Can tolerate missed messages but not afford to process duplicates	Start the producer against the Confluent cluster before the consumer or they are stopped and repointed at the same time. The consumer can be started from the latest offset or a translated offset.	Start the consumer against the Confluent cluster before the producer or they are stopped and repointed at the same time. The consumer can be started from the latest offset on the Confluent cluster.
Cannot process duplicates or miss messages	Consumer group offset syncing or translation between clusters can simplify the migration process and mitigate client downtime. Otherwise, the client migration sequence must be carefully orchestrated to ensure the data processing requirements are met.	The client migration sequence must be carefully orchestrated to ensure the data processing requirements are met.

Review Potential Security Changes

Often in Apache Kafka deployments, Kafka security settings are absent as administrators commonly depend on their firewall or VPC security groups. However, when it comes to a cloud environment, implementing additional security controls is necessary to keep the workloads safe.

Encryption

Storage level encryption of data at-rest is provided to all Confluent Cloud clusters by default. Confluent Cloud also provides optional support for self-managed encryption keys, also known as bring-your-own-key (BYOK) encryption, when you create Dedicated Kafka clusters in Confluent Cloud. This option is preferable for organizations that want to use their own encryption key to encrypt data at-rest or require the option to disable access to the encrypted volumes by Confluent. BYOK provides a greater degree of privacy and data integrity, which is often required for compliance by many regulated industries such as government, healthcare, and financial services.

In addition to data at-rest encryption, network level encryption is automatically enforced for data in-motion. In Confluent Cloud, all network traffic between clients and brokers is encrypted with TLS 1.2 and this cannot be disabled. Depending on your client deployment, you may need to import the Let's Encrypt root CA certificates into the system trust store or cacerts file, or upgrade the underlying operating system or JDK to successfully establish the TLS handshake[3]. This is especially pertinent to non-Java clients or for Java users with a version less than 7u151 or 8u141.

Authorization and Authentication

Confluent Cloud offers two cloud-native authentication mechanisms - Confluent OAuth and/or API Keys - to control access to Confluent Cloud components and resources. You can manage client access and user access to Confluent Cloud by associating these with individual service accounts and user accounts, respectively. Permissions can be specified using ACLs and/or role bindings tied to specific service or user accounts. Group mappings can be leveraged to define a set of rules that map user groups in your SSO identity provider to Confluent Cloud RBAC roles.

Of these two authentication mechanisms, OAuth is the most commonly chosen since it:

- Provides a central authority for all client authentication
- Reduces operational burden associated with credential management
- Is commonly leveraged in cloud computing

Security Differences Between Apache Kafka and Confluent Cloud

Security dimensions		Apache Kafka	Confluent Cloud
Authentication	SASL_PLAIN	☑	☑
	SASL_SCRAM	☑	—
	GSSAPI (Kerberos)	☑	—
	OAuth	☑	☑
	mTLS	☑	☑
Authorization	ACLs	☑	☑
	RBAC	—	☑
Network Encryption	Plaintext (No encryption)	☑	—
	TLS	☑	☑
Audit	Audit Logs	—	☑

[3]For Confluent customers, check out our Support article for more instructions.

Select Network Connectivity

Confluent Cloud supports public internet connectivity as well as private networking solutions. The table below specifies the available networking options for each cluster type.

		Confluent Cloud cluster type			
Supported network type		**Basic**	**Standard**	**Enterprise** (AWS and Azure Only)	**Dedicated**
Public internet		☑	☑	–	☑
Private networking	Private Link connections	–	–	☑	☑
	VPC/VNet peering	–	–	–	☑
	AWS Transit Gateway	–	–	–	☑

Secure Public Endpoints

Confluent Cloud clusters with secure internet endpoints are protected by a proxy layer that prevents methods of DoS, DDoS, syn flooding, and other network-level attacks. For Confluent Cloud clusters with public connectivity, you can use public egress IP addresses to communicate with external resources (such as data sources and sinks for managed connectors) over the public internet. For details, see Use Public Egress IP Addresses on Confluent Cloud for Connectors and Cluster Linking.

Private Networking

Private networking further reduces potential security threats by only allowing access to your cluster from the private cloud service provider networks (VPCs/VNets) that you specify, and not allowing access to your cluster from the public internet. To access your cluster, someone must first be given access to your private network. Private networking requires managing the peered or linked networks to ensure all clients and developers can access Confluent Cloud. Be sure to consider the networking of your existing VPCs/VNets to establish connectivity between your clients and Confluent. To learn more about leveraging private connectivity with Confluent Cloud, watch Confluent Cloud Networking Basics and Apache Kafka Connectivity.

Select Confluent Cloud Cluster Type

When selecting a cluster on Confluent Cloud, there are several different cluster types to choose between. Your Confluent Account team is available to help you choose the cluster type that best fits your workload.

There are 4 cluster types available:

Basic: designed for experimentation, early development, and basic use cases. These have fixed capacity and are provisioned with enough reserved capacity to support them.
Standard: designed for production-ready features and functionality. These have fixed capacity and are provisioned with enough reserved capacity to support them.
Enterprise: designed for production-ready functionality that requires private endpoint networking capabilities. These are elastic and automatically scale up and down to meet the needs of the workload up to a fixed capacity limit.
Dedicated: designed for critical production workloads with high traffic or private networking requirements. These clusters elastically scale at the request of the user to support the largest workloads.

For more information regarding the different cluster configurations, please refer to this documentation. It is essential to consider all dimensions of a cluster's capacity to ensure that your workload will successfully run on the chosen cluster type.

Select Optimal Data Replication Tool (Optional)

If you need to migrate data from the Apache Kafka cluster to Confluent Cloud, then you'll need to choose the appropriate migration tool. Today, there are three primary migration tools available to migrate data from Apache Kafka to Confluent Cloud:

1. Confluent Cluster Linking (recommended)
2. Confluent Replicator
3. MirrorMaker 2.0 (MM2)

While it is most common for a single replication tool to be used during the migration, it is possible to use multiple replication tools where required. For example, if you want to repartition a subset of the topics, you could use a mix of Confluent Cluster Linking and Confluent Replicator.

An important feature to evaluate for each tool is how consumer offsets are handled. When consumers move from the source cluster to Confluent Cloud, you will need to ensure consumers resume reading from the offset that aligns with your processing requirements (that is, if you are replicating data). Resuming from the appropriate consumer offset can simplify the migration of the consumer client and essentially allow a consumer to continue processing as if the Kafka cluster has not changed.

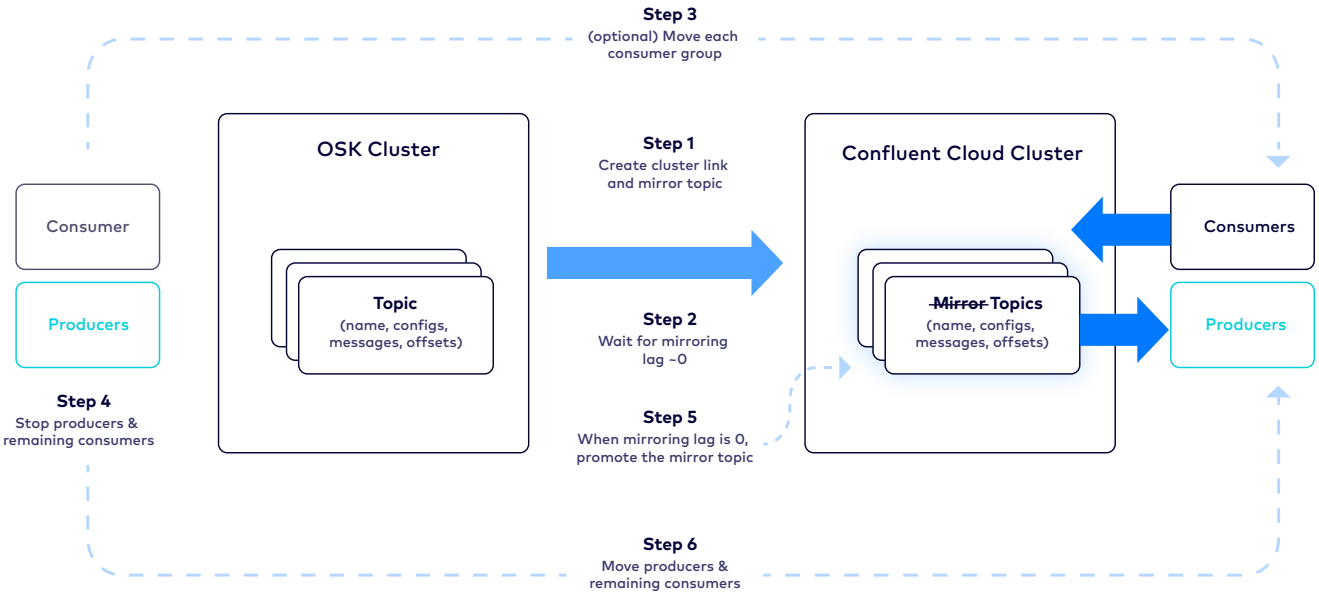
Confluent Cluster Linking

Cluster Linking on Confluent Cloud is a fully managed service for replicating data from one cluster to another. Programmatically, it creates exact copies of your topics and keeps data in-sync across clusters. Cluster Linking is a powerful geo-replication technology fit for a variety of use cases, including data migrations.

Cluster Linking creates an identical copy of your topics on the destination cluster, making it easy to move to the Confluent cluster with little downtime and no data loss. Cluster Linking can:

- Automatically create matching "mirror" topics with the same configurations, so you don't have to manually recreate your topics in the Confluent cluster
- Sync all historical data (optional) and newly written data from the existing topics to the mirror topics
- Sync your consumer group offsets for any client library allowing your consumers to pick up exactly where they left off when they were stopped on the original cluster, without missing any messages or consuming any duplicates
- Automatically scale to meet throughput requirements and limit replication lag
- Offer flexibility to independently move just the consumers from the origin cluster to the Confluent cluster

Due to its ease of use and rich feature set, Cluster Linking is the **recommended** data replication tool for any migration it can be applied to. However, there are situations when another tool needs to be used. Be sure to review Cluster Linking's limitations before selecting it as your replication tool for the migration effort.



Advantages	Limitations
✓ Confluent supported	✗ Cannot be used to re-partition or rename topics
✓ Easy to use	✗ Cannot be used to newly encrypt message fields during the replication process
✓ Fully managed - no need to manage or tune data flows	✗ Some source/destination cluster pairings may not be supported. See supported cluster types
✓ Byte-for-byte topic replication, giving a guarantee of no duplicate messages and keeping the Confluent cluster fully consistent with the origin cluster	
✓ Can start replication from any place in the source topics	
✓ Consumer offset syncing for any client library	
✓ Does not require provisioning any extra software components or infrastructure	
✓ Maintains encryption	

Cluster Link Consumer Offset Sync

The byte-for-byte nature of Cluster Linking data replication keeps the message offsets consistent between clusters, offering the added flexibility of migrating the consumer clients to the destination cluster without extra tooling. This is accomplished by leveraging the link's consumer offset sync functionality that replicates consumer group offsets to the destination cluster. When a consumer starts against the Confluent cluster, it picks up from the last offset that was committed on the source cluster. Enabling consumer offset syncing is a matter of setting a few configuration parameters as part of the link deployment, all of which are detailed in the Configure and Deploy Replication Tools (Optional) section.

An important thing to note with Cluster Linking, like all of the tools in this list, is that it is an asynchronous data replication process, and the syncing of the consumer offsets is no different. The consumer offsets are written to the destination cluster `every consumer.offset.sync.ms` milliseconds—a configurable cluster link parameter with a default of 30 seconds. This configuration defines the minimum time period that a consumer should wait between its last offset commit on the origin cluster and the first time it begins consuming data on the destination cluster. If less time than this setting has elapsed during the consumer migration—for example, if the consumer offset sync parameter is at 30 seconds but the consumer disconnects from the origin cluster and reconnects to the Confluent cluster in just 5 seconds—then there is a chance the consumer will consume duplicate messages. If your consumer groups need less than 30 seconds of downtime when switching from the origin cluster to the Confluent cluster, then set this parameter to a lower number for a more frequent consumer offset sync.

Case study

SAS: A much easier migration thanks to Cluster Linking

To enable real-time data for their Customer Intelligence 360 suite, SAS wanted to deploy Kafka on their private cloud environment based on AWS EKS. But deploying Kafka on Kubernetes is a complex, challenging operation. Justin Dempsey, Senior Manager, SAS Cloud says, "Our transformation to a cloud-native, agile SaaS solution required a large-scale migration from open source Apache Kafka, which was complex and challenging to self-support."

Cluster Linking, a solution to easily link Confluent clusters together to form a highly available, consistent, and real-time bridge across environments, was a critical aspect of the project because it streamlined the anticipated migration from the old Kafka infrastructure to the new. During the cross-divisional effort between SAS Cloud, R&D, and other SAS teams, they developed an approach using Cluster Linking to enable them to have effectively zero downtime. Dempsey's team was able to leverage Cluster Linking to seamlessly migrate existing clusters, enabling the team to build a private cloud Kafka service on their Kubernetes distribution of choice in order to properly power their customer 360 use cases.

"Cluster Linking played a critical role in making the shift to Confluent for Kubernetes and our more cloud-native solutions a success, with no downtime and minimal dependencies."

— Justin Dempsey, Senior Manager, SAS Cloud

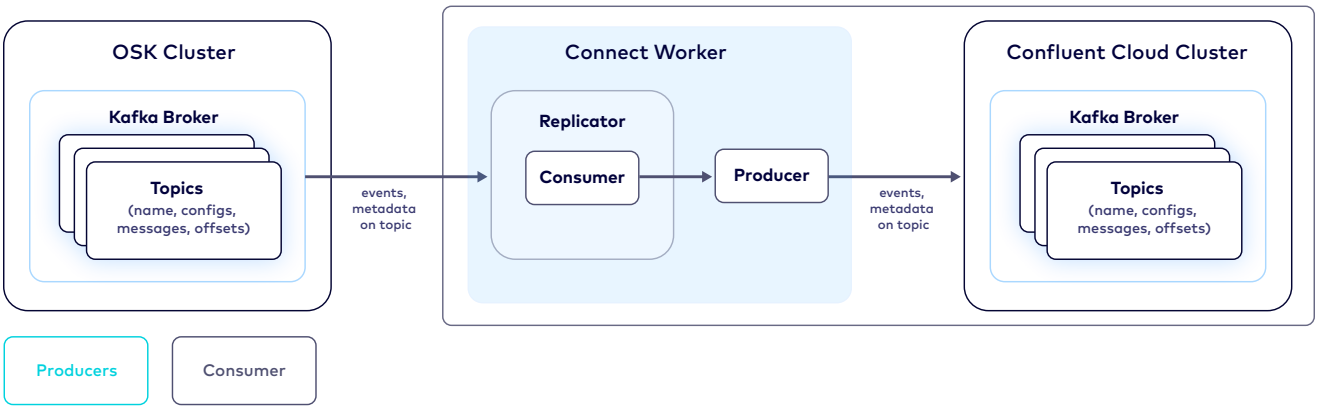
Read more

Confluent Replicator

Confluent Replicator is recommended if you have chosen to repartition or reconfigure topics as a part of the migration, or if Cluster Linking is not supported for the source/destination cluster pairing (source cluster version, destination cluster type, or networking restrictions).

Replicator allows you to reliably replicate topics from one Kafka cluster to another. In addition to copying the messages, Replicator will automatically create topics as needed, preserving the source cluster's topic configuration. This includes preserving the number of partitions, the replication factor, and any configuration overrides specified for individual topics. When topic configuration changes are made in the source cluster, Replicator will automatically reconfigure those on the destination cluster.

Replicator can be run as an executable or a connector, but is best run as a connector for resiliency and ease of operation. When Replicator is run as a connector, it is recommended to be deployed nearest to the destination to optimize performance. Replicator doesn't support exactly once data replication, which means if the connector task restarts there may be some duplicate records in the destination Kafka topic. Replicator essentially works as a consumer from the source cluster and a producer to the destination cluster.



Advantages	Limitations
✓ Confluent developed, supported and battle tested with hundreds of Enterprises since 2016	✗ More complex setup than Cluster Linking
✓ Allows topics to be reconfigured on the destination cluster (i.e. renamed, repartitioned, etc.)	✗ Needs a separate self-managed Connect cluster unless deployed as an executable
✓ Allows topics to be reconfigured on the destination cluster (i.e. renamed, repartitioned, etc.)	✗ No offset translation for non-Java clients, yet this can be accomplished using the kafka-consumer-groups CLI or the offsetsForTimes method
✓ Supports offset translation for Java clients	✗ Does not support exactly-once data replication
✓ Maintains encryption	

Repartition or Reconfiguring Topics

Replicator supports configurations that disable topic configuration syncing and auto topic creation, allowing for topic repartitioning and reconfiguration. Make the following Replicator configuration changes to support repartitioning or reconfiguration of the destination topics. Keep in mind that repartitioning topics with keys will impact the key to partition alignment when the data is replicated to the new cluster.

Config	Default Value	Updated Value
topic.auto.create	true	false
topic.preserve.partitions	true	false
topic.config.sync	true	false

Replicator Consumer Offset Translation

When replicating data with Confluent Replicator, there are two methods of syncing consumer offsets to the Confluent cluster:

- Via the in-built consumer offset translation using the Consumer Timestamps Interceptor with Java clients
- Via the `kafka-consumer-groups` CLI or the `offsetsForTimes` method for non-Java clients

Replicator can replicate consumer offsets for Java clients through a process called consumer offset translation. To facilitate the translation, the consumer client must be configured with an interceptor called the Consumer Timestamps Interceptor. This interceptor preserves metadata of consumed messages, and the consumer timestamp information is preserved in a Kafka topic, `__consumer_timestamps`, located in the source cluster.

For non-Java clients, you can translate consumer offsets using the kafka-consumer-groups CLI or the offsetsForTimes method. These solutions will direct the consumer group to start from the offset closest to the provided timestamp, which can be a timestamp that is just before the time the consumer was stopped in an effort to limit the number of duplicates that will be consumed.

MirrorMaker 2.0

MirrorMaker 2.0 (MM2) is best suited for situations where Cluster Linking and Replicator are not a good fit. MirrorMaker 2.0 can be used in any scenario where Replicator would be used, however, it is more challenging to deploy and troubleshoot due to limited documentation. Due to this, MirrorMaker 2.0 is only recommended to be used for the limited use cases where Cluster Linking and Replicator cannot be used.

Advantages	Limitations
✓ Allows topics to be reconfigured on the destination cluster	✗ More complex setup than Cluster Linking
✓ Supports offset translation for all clients	✗ Needs a separate self-managed Connect cluster unless deployed as an executable
✓ Allows topics to be reconfigured on the destination cluster (i.e. renamed, repartitioned, etc.)	✗ Limited documentation and examples
✓ Supports exactly-once data replication (KIP-618)	✗ For support, Confluent requires this to be run in "Dedicated" mode
✓ Maintains encryption	✗ Many internal topics created on both source and destination clusters that need clean-up post-migration

MirrorMaker 2.0 Consumer Offset Syncing

MirrorMaker 2.0 supports consumer offset syncing by leveraging the MirrorClient to translate the consumer offsets from the source to the destination cluster, and periodically synchronizing the translated offsets to the `__consumer_offsets topic` in the destination cluster. This feature can be enabled by setting both `sync.group.offsets.enabled` and `emit.checkpoints.enabled` to true. These parameters cause the `MirrorCheckpointTask` to sync the selected and translated consumer group offsets to the target cluster. The frequency of the offset sync is the same as the frequency of emitting checkpoints.

MirrorMaker 2.0 will only sync offsets for the consumers that are inactive in the destination cluster, and if the high watermark of the consumer offsets at the destination cluster is lower than the offsets at the source cluster.

Select Optimal Schema Replication Tool (Optional)

To guarantee data quality in the Kafka cluster, it is recommended to use Confluent's fully managed Schema Registry to support the management and evolution of message schemas. This seamlessly integrates with Confluent clusters, connectors, Flink compute pools, and ksqlDB applications.

In Confluent Cloud, clusters and resources are grouped into Confluent Environments. Each Environment can only have one Schema Registry instance, which is used by all of the connectors, Flink compute pools, and ksqlDB applications in that Environment.

If your existing Kafka clients are using a Schema Registry when writing messages to the topics, consider the following solutions to migrate the schemas to the Confluent Cloud Schema Registry.

Confluent Schema Linking

Confluent offers Schema Linking as a fully managed service for replicating schemas between Schema Registries. Schema Linking is the direct complement to Cluster Linking. Schema Linking ensures the schema IDs are kept the same between the two Schema Registries, which allows the replicated messages to be effectively deserialized by consumers referencing the Confluent Schema Registry URL.

Schema Linking introduced the concepts of a schema exporter and schema context. A schema exporter resides within a Schema Registry cluster and can be used to replicate schemas between two Schema Registry clusters. A schema context is essentially a logical grouping of subject names and schema IDs that can be thought of as a separate "sub-registry" in the same Schema Registry cluster. Schema exporters run on the source cluster and sync schemas to the target context in the destination.

Schema Linking requires the source Schema Registry to be Confluent Platform version 7.0+ since the exporter runs on the source.

Replicator Schema Translation

Another available schema migration solution is the schema translation functionality of Confluent Replicator. Replicator schema migration supports replicating an entire Schema Registry to another (empty) Schema Registry cluster or subject in IMPORT mode, preserving all schema ID and version information. This is accomplished by replicating the `_schemas` topic and using the `ByteArrayConverter`. When using schema translation, there can be an impact on the data replication performance.

Confluent Replicator schema translation functionality can also be used on its own with another data replication tool.

Schema Registry API

If you are using a Schema Registry that is not Confluent Schema Registry for your Kafka workloads, the most flexible schema migration solution is to use the Schema Registry API on both the source and the destination.

Some of the available open source Schema Registries have Confluent compatible endpoints. If the clients already use the Confluent compatible endpoints and the Confluent supported serdes libraries, use the REST API to register the same schema with the same ID in the Confluent Cloud Schema Registry. Doing so will allow the consumers to deserialize the replicated messages. This can be done by first setting the cluster or subject in IMPORT mode and then registering a new schema by explicitly providing the ID. The ccloud-schema-exporter open source tool can be used as a reference for this process.

Other Schema Replication Solutions

If your existing Schema Registry cluster does not have Confluent compatible endpoints and your clients are using serdes libraries specific to that service, a custom-built solution will be required to replicate both the data and schemas. This is because the magic byte and schema identifier in the payload used by the Confluent serdes libraries to associate a message with the schema ID used to serialize it, is likely a different format than what is used by the non-Confluent libraries.

There are examples of custom-built solutions that have been used to successfully migrate to Confluent Cloud from clusters with data serialized by clients that were integrated with non-Confluent Schema Registries. However, these solutions also require updates to the client configuration to use the Confluent schema-aware serdes libraries. Please contact Confluent Professional Services for assistance with custom schema migration tooling.

Set Up

After defining a clear migration plan, you are ready to provision your Confluent Cloud clusters. It is recommended to iteratively set up clusters. For example, when in the testing phase, wait to set up the production clusters until the tests are successful. This will allow you to more easily adjust your cluster configurations where needed and also reduce migration costs. Confluent recommends using the Confluent Terraform provider to deploy and manage Confluent Cloud infrastructure so that you can build, change, and version your cloud data infrastructure in a safe and efficient way.

Prepare the Cluster

When creating a cluster, the following configurations must be selected and cannot be changed:

- Cluster Type[4]
- Cluster Networking
- Cluster Region
- Single or Multi-Zone Selection
- Bring-your-own-key (BYOK) encryption[5]

After the cluster has been created, create the following:

- A unique Service Account for each client with appropriately assigned key pairs or OAuth identity pools, and RBAC roles following the concept of least privileges
- User accounts with appropriate RBAC roles
- Topics and schemas - either manually[6] or automatically via a replication tool if using one

Next, validate network connectivity between source and destination clusters (if you are replicating data/schemas), and between clients and the Confluent Cloud resources.
If you plan to replicate data as part of the cluster migration process, set up the chosen replication tool to begin replicating the data to Confluent Cloud.

Configure and Deploy Cluster Linking

For a step-by-step walkthrough of how to set up and configure Cluster Linking, check out the Cluster Linking Migration Tutorial.

For Consumer Offset Syncing, enabling this feature is easy and a matter of setting a few configuration parameters. Lower the `consumer.offset.sync.ms` configuration to be less than the amount of time it takes for the consumer to be repointed to the Confluent Cloud cluster and restarted. This will ensure that the consumer offsets have synced to Confluent Cloud before the consumer begins processing against the Confluent cluster. Consumer offset sync can be set as frequently as every 1 second[7].

[4]Basic clusters can be promoted to Standard clusters after creation.
[5]BYOK is only available for Dedicated clusters.
[6]Using `auto.create.topics=true` is never recommended for production. If you choose to use this during the migration, be sure to set it to false after migrating.
[7]Reducing the offset sync comes at a higher cost since consumer offset sync is billed as a part of Cluster Linking throughput. The cost is proportional to the number of consumer groups and the number of partitions.

Once Cluster Linking is replicating data, wait for the replication lag to be at or near 0 before beginning the migration process to limit the amount of time required to perform the necessary steps. Cluster Linking replication can be monitored via cluster_link_mirror_topic_offset_lag in the Confluent Cloud Metrics, or using the confluent CLI.

Configure and Deploy Confluent Replicator

For a step-by-step walkthrough of how to deploy Confluent Replicator, check out the Confluent Replicator Tutorial. The recommended method of deploying a self-managed Replicator instance is via Confluent for Kubernetes, which includes a set of detailed examples to guide customer deployments.

Rather than deploying Replicator as a self-managed connector, it can also be deployed in Confluent Cloud as a custom connector, where Confluent manages the infrastructure to run the connector on behalf of the customer.

For Replicator Consumer Offset Translation, be sure to configure Timestamp Preservation in the consumer client and enable Replicator offset translation. Once Replicator is running, wait for the replication lag to be at or near 0 before migrating. Replicator lag can be monitored by using the Consumer Group Command tool (`kafka-consumer-groups`), the available JMX metrics, or using Confluent Control Center with the Replicator monitoring extension.

Configure and Deploy MirrorMaker 2.0

For an overview of the different ways to deploy MirrorMaker 2, check out the KIP and Kafka github repository.

For MirrorMaker 2 consumer offset syncing, the following configuration parameter must be set `sync.group.offsets.enabled=true`. Monitoring MirrorMaker 2 replication lag can be done using the JMX metrics `replication-latency-ms` and `record-age-ms` from `kafka.connect.mirror:type=MirrorSourceConnector,target=([-.w]+),topic=([-.w]+),partition=([0-9]+)`.

Configure and Deploy the Schema Replication Tool (Optional)

For tutorials on how to set up Schema Linking, schema replication with Replicator or schema replication with the REST API, see the following tutorials and examples:

- Schema Linking Quick Start Tutorial
- Migrate Schemas with Replicator
- ccloud-schema-exporter

Prepare the Clients

All clients will need to update their bootstrap server and security configurations to connect to the Confluent Cloud cluster. If integrating with Confluent Cloud Schema Registry, the Schema Registry URL and security configurations will also be required. As with any Kafka cluster, updating these configurations requires restarting the client. The most common approaches for updating the client credentials are:

1. If the clients are deployed in Kubernetes, then the bootstrap server and credentials should be passed to the Kafka clients as an environment variable or Kubernetes secret. When starting client pods, the configurations will be automatically passed into all client configurations that reference the environment variable or secret.
2. If the Kafka clients are not run on Kubernetes, it is best practice to store the bootstrap server of the active cluster in a Service Registry or Service Discovery tool, like Hashicorp Consul. When a client starts up, it fetches the bootstrap server from the service registry. To trigger the migration, change the Apache Kafka bootstrap server in the service registry to that of the Confluent Cloud cluster. Then, when your clients restart, they will bootstrap to the Confluent Cloud cluster.
3. If the Kafka clients integrate with Schema Registry, they should get the URL of the Confluent Cloud Schema Registry in the same way as they get the Confluent Cloud bootstrap server.
4. If the clients will use OAuth to authenticate with Confluent, `extension_logicalCluster` must be updated to the Confluent Cloud resource IDs. Like the bootstrap server, this can be passed in from Kubernetes or Terraform, or stored in a Service Registry or Service Discovery tool.
5. If the clients will use API keys, then they need a set of API keys for the Confluent Cloud Kafka cluster and Schema Registry cluster (if applicable). Clients should fetch the active API key pair from a secret manager like Hashicorp Vault or AWS Secrets Manager.

For more client configuration information, check out our documentation Client Producer and Consumer Configuration Recommendations for Confluent Cloud.

Benchmark

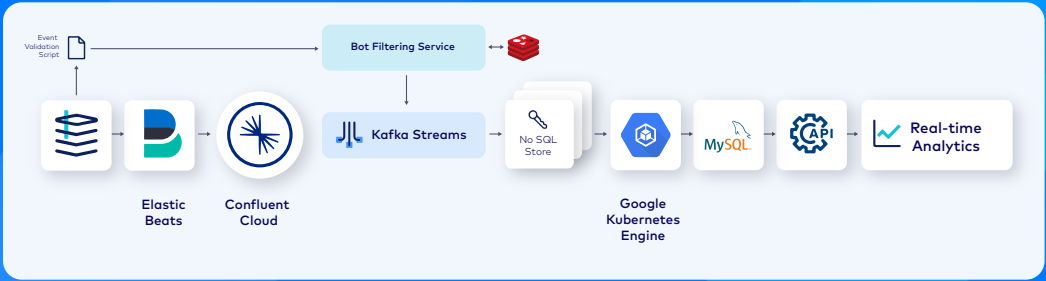
Benchmarking is an important part of the migration process and should be done for all migration use cases. During the planning phase, you should identify key benchmarking criteria for the workloads to ensure consistency when moving to Confluent Cloud. After setting up a cluster, benchmarking tests should be run to validate the cluster set up. Performing benchmark testing will confirm the Confluent Cloud cluster is appropriately configured for your use cases. For an overview of how to benchmark clients, see our documentation.

Another commonly used benchmarking framework is OpenMessaging, which is a suite of tools that simplifies benchmarking of distributed messaging systems in the cloud. For example configurations, see the following repository.

Case study

BigCommerce: Moving 1.6B events per day with zero downtime, zero data loss, and zero disruption

BigCommerce migrated from open source Kafka to Confluent Cloud because they needed a fully managed, cloud-agnostic platform with pre-built enterprise features to free up their engineering time for managing and scaling Kafka. For them, the migration strategy had to be flexible enough so that operations would not be shut down. The team meticulously planned the migration with the help of Confluent's expertise to design the partition strategy and migration solution. This was done to prepare for future needs and traffic variations for production use cases. Eventually, the team decided to pump their critical workloads into both their open source Kafka model and the new Confluent model for a period of time. That allowed them to scale in a test environment, starting with a low volume of data, ensuring things would work smoothly, and then seamlessly switch over completely. They then proceeded with lag reporting integration, batch workload migration, and Kafka stream optimization to wrap up the migration.



Real-time analytics and insights for merchants' architecture

With such a tight migration strategy, in just five months, BigCommerce had migrated 1.6 billion events per day, 22 broker nodes, three clusters, and 15 TB of storage data to Confluent Cloud. They experienced zero downtime, zero data loss, and zero disruption to the merchant experience. Now, they can seamlessly scale data with low operational overhead and maintenance, saving 20+ hours a week in Kafka management and infrastructure costs from no longer having to overprovision clusters to accommodate seasonal traffic spikes.

Read more

Migrate

Now that you've developed a detailed plan, provisioned the cluster and replication tooling (if required), and benchmarked the clients against the Confluent Cloud cluster, it is time to migrate the clients! Two main client migration patterns are typically followed:

- Migrating an entire cluster and all associated workloads at once: all-at-once
- Migrating workloads in phases over a longer period of time: phased approach

The most common pattern to follow when performing the client migration is the all-at-once pattern; either with or without replicating data in topics. The "all-at-once" pattern involves moving all clients in one motion. This pattern is generally recommended since it:

- Avoids the need to untangle complex client interdependencies
- Delivers the highest ROI on Confluent Cloud
- Reduces the overall migration timeline

We will lay out the steps required for an all-at-once migration pattern, with and without replicating topic data.

Note: If choosing a phased migration approach, you can follow steps similar to the all-at-once pattern for the subset of clients and topics that are being migrated in each phase.

Client Migration Considerations

Regardless of whether you are replicating data, it is important to be mindful of any client interdependencies that may require some clients to be restarted before others.

If a Consumer is dependent on a Producer...	The clients should be migrated appropriately to prevent situations where the consumer may still be pointing to the origin cluster and not see new data written by the producer to the Confluent cluster. In some cases, this may mean migrating the consumer before the producer.
If a client is both a Producer and a Consumer...	It is best to review migration strategies where consumer offsets are replicated/translated. Offset replication/translation ensures that the consumer will not miss processing any messages during the migration. Without this, there is no way to guarantee all data has been consumed by the client before stopping it.

The table below provides examples of how data processing requirements will influence how the clients should be restarted.

Data processing requirements	How clients should be restarted
Consumers can miss data and/or process duplicates[1]	All clients can be stopped and restarted at the same time against Confluent Cloud, regardless of the consumer configuration
Consumers can miss data but not process duplicates	All clients can be stopped and restarted at the same time against Confluent Cloud, and consumers should be configured with `auto.offset.reset=latest`
Consumers cannot miss data but can process duplicates[1]	All clients can be stopped and restarted at the same time against Confluent Cloud, and consumers should be configured with `auto.offset.reset=earliest`

The migration process can be much more flexible for cases where consumers do not have these requirements and there is no particular order with which the clients should be restarted.

Migrate without Replicating Topic Data

A migration that does not require data to be replicated to the Confluent cluster is the simplest effort. There are 3 common patterns that can be followed.

Client Migration Pattern: Restart All At Once

If consumers can afford to miss messages (in some cases) and no client interdependencies are dictating the order clients restart, then simply restart all clients to redeploy to the Confluent Cloud cluster. Since the client's bootstrap server and security configurations were updated in the setup phase, when the clients restart they will point to the Confluent cluster and begin processing.

[1]If data is not being replicated to the destination cluster, processing duplicate messages will not be a concern.

Client Migration Pattern: Stop-Wait-Restart

The Stop-Wait-Restart client migration pattern is the most commonly used sequence of steps when data is not migrated to the Confluent cluster. This pattern will assume:

- Consumers should not miss data or process many duplicates
- Client downtime is acceptable

It is recommended to run this pattern during a pre-planned maintenance window to reduce or eliminate the impact of the client downtime. Advanced operators can move individual consumers (and subsequently the relevant producers) as soon as the lag hits 0 to reduce the downtime. However, this can increase the risk of restarting clients out of the required order (if applicable) and should be done with caution.

Stop-Wait-Restart Steps:

1. Stop all producers on the origin cluster
2. Wait until the consumer lag for all consumers is 0 on the origin cluster
3. Restart consumers against the Confluent cluster. Since there is no data on the Confluent cluster, the consumer configuration **`auto.offset.reset`** is not of concern
4. Restart the producers against the Confluent cluster

Note: Depending on processing and downtime requirements, if producers are restarted against the Confluent cluster before consumers, be sure the consumers are configured with **`auto.offset.reset=earliest`** *so no data is missed.*

Client Migration Pattern: Blue/Green

If you are leveraging a blue/green deployment strategy to avoid client downtime, then you can point any upstream and downstream services to use the clients running against Confluent Cloud following validation.

Migrate with Replicating Topic Data

When the migration requires historical data to be replicated to the Confluent cluster, there are a few common options for how to migrate the clients. Each client migration pattern has its trade-offs, so it is important to review each and determine which is best for your migration effort. Please note that different replication tools have different impacts on the sequence of steps for each pattern. See the Select Optimal Data Replication Tool section for more details on the differences between the tools.

Pattern	Assumptions
Stop-Wait-Restart *Commonly used for Replicator with non-Java client use cases	• Consumers should not miss data or process many duplicates • Client downtime is acceptable • No consumer offset replication or translation is in use
Stop-Restart-Repeat	• Consumers should not miss any messages • Minimal client downtime • No duplicate processing (Cluster Linking) • Minimal duplicate processing (Replicator/MirrorMaker 2.0) • Flexibility to migrate consumers without first stopping producers

Client Migration Pattern: Stop-Wait-Restart

The Stop-Wait-Restart client migration pattern is the most straightforward, the simplest to execute, and all three data replication tools can be used. The sequence is nearly identical to the pattern when data is not replicated.

This pattern will assume:

- Consumers should not miss data or process many duplicates
- Client downtime is acceptable
- No consumer offset replication or translation is in use

It is recommended to run this pattern during a pre-planned maintenance window to reduce or eliminate the impact of the client downtime. Advanced operators can move individual consumers (and subsequently the relevant producers) as soon as the lag hits 0 to reduce the downtime. However, this can increase the risk of restarting clients out of the required order (if applicable) and should be done with caution.

Client Migration Pattern: Stop-Restart-Repeat

The Stop-Restart-Repeat client migration pattern is common in scenarios where clients require as little downtime as possible and a consumer offset replication/translation strategy is in use. Due to the differences in each of the common data replication tools, the steps for this pattern differ depending on the chosen tool.

This pattern will assume:

- Consumers do not miss any messages
- Minimal client downtime
- No duplicate processing (Cluster Linking)
- Minimal duplicate processing (Replicator/MirrorMaker 2.0)
- Flexibility to migrate consumers without first stopping producers

Stop-Wait-Restart Steps:

1. Stop all producers on the origin cluster
2. Wait until:
    - All data is replicated to the Confluent cluster
    - The consumer lag for all consumers is 0 on the origin cluster
3. Restart consumers against the Confluent cluster with `auto.offset.reset=latest`. This configuration will ensure the consumer reads from the log end offset of the topic
4. Restart the producers against the Confluent cluster
5. If using Cluster Linking, use the promote command to convert the mirror topics to writable topics prior to restarting the producers

Cluster Linking Stop-Restart-Repeat Pattern Steps

More detailed instructions can be found in our Standard Migration documentation

1. Ensure consumer offset syncing has been configured for the Cluster Link
2. Wait for the replication lag of the Cluster Link to be at or near 0
3. Stop all consumers on the origin cluster
    - At this point, it is recommended to disable consumer offset syncing for all consumer groups or the target set of consumer groups with `consumer.offset.group.filters` to prevent potential offset over-writes in the Confluent cluster
4. Restart the consumers against the Confluent Cloud cluster
    - These will start from where they left off on the origin cluster since consumer offsets have been synced. Be sure to review the `consumer.offset.sync.ms` configuration parameter so it aligns with client restart expectations
5. Stop all producers on the origin cluster
6. Promote the mirror topics
    - The promote command promotes mirror topics to be writable topics once all data has synced from the source topic
7. Immediately after promoting the mirror topics, restart the producers
    - The promote command is nearly instantaneous. The only time incurred will be the time to replicate the remaining offsets for the topic. Since Step 2 is to wait for the replication lag to be near 0, there should be little time to wait

Replicator (Java clients) / MirrorMaker 2.0 Stop-Restart-Repeat Pattern Steps

1. Ensure consumer offset syncing/translation has been configured. See the Select Optimal Data Replication Tool section for details
2. Wait for the replication lag to be near 0
3. Stop all consumers on the origin cluster
4. Restart the consumers against the Confluent cluster. These will start from where they off on the origin cluster so long as:
    - For Replicator, the consumers are configured with `interceptor.classes=io.confluent.connect.replicator.offsets.ConsumerTimestampsInterceptor`
    - For MirrorMaker 2.0, the deployment is configured with `sync.group.offsets.enabled=true`
5. Stop all producers on the origin cluster
6. Restart producers against the Confluent cluster
    - If order matters, be sure that replication lag is 0 before restarting the producers

Replicator Steps (non-Java clients) Stop-Restart-Repeat Pattern Steps

1. Wait for the replication lag to be near 0
2. Stop all consumers on the origin cluster
    - Note the timestamp at which the consumers were stopped
3. Using the timestamp from Step 2, update the consumer group offset for each consumer on the Confluent Cloud cluster to align with the timestamp that the consumer was stopped
    - This can be done with the `kafka-consumer-groups` CLI or `offsetsForTimes()` and `seek()` methods
    - It is best to start at an offset that aligns with a timestamp a few minutes before the consumers are stopped to be sure that no data is missed
4. Restart the consumers again the Confluent cluster
5. Stop all producers on the origin cluster
6. Restart producers against the Confluent cluster
    - If order matters, be sure that replication lag is 0 before restarting the producers

Client Migration Pattern: Blue/Green

Blue/Green deployments can be achieved with data replication, however, it can require client downtime to ensure the data is consistent between the clusters and to mitigate duplicate writes. This setup can be quite nuanced and complex, so it is recommended to work with Confluent Professional Services if your migration efforts will require this setup.

Migrating Schema Registry

If migrating a Schema Registry cluster to Confluent Cloud Schema Registry, review the links in the Select Optimal Schema Replication Tooling section to appropriately set up your selected schema replication tool. In most use cases, the clients are already configured to use the Confluent serializers and deserializers with the origin Kafka cluster. Confluent schema aware serializers and deserializers use a specific wire format so the deserializer is able to interpret which schema to use when reading the serialized bytes of a topic. Therefore, it is required to keep the schema IDs the same between the Schema Registry clusters when the data replication tool replicates the serialized bytes between clusters. This is relevant to Cluster Linking or using ByteArrayConverters with Replicator and MirrorMaker 2.0.

If a different Schema Registry cluster and associated serdes libraries were used with the origin Kafka cluster, additional logic must be included in the data and schema replication solution.

Validate

After migrating to Confluent Cloud, it is recommended to perform a detailed validation to ensure that the migrated state meets the requirements and goals defined during the planning phase. Once everything is validated, it is time to clean up any tooling and infrastructure that was provisioned to support the migration, as well as the original Kafka cluster.

Ready to get started?

And now you're ready to migrate your Kafka applications! Hopefully this provides a useful guide to how you can start approaching your open-source Kafka migrations to Confluent Cloud.

With Confluent, you can free your teams to move up the stack and focus on developing real-time applications and streaming pipelines rather than managing day-to-day operations. Organizations that use Confluent have been able to save 60% on their Kafka TCO and move their streaming use cases into production over 6 months faster than with self-managing open source Kafka. And all of this is underpinned by our 99.99% uptime SLA, supported by the experts with 1M+ hours of Kafka experience and responsible for over 80% of Kafka commits. Plus, Confluent offers hands-on implementation services, interactive workshops, and technical advisors to assist you on your data streaming journey.

To get started with your Kafka to Confluent migration, contact us and one of our Kafka experts will be in touch. You can also check out Confluent Professional Services (PS) and our Partner Ecosystem as there may be additional incentives for qualifying organizations that can help offset the migration costs and drive a lower TCO.

For organizations interested in jumpstarting their data streaming journeys away from legacy data infrastructure, the Confluent Migration Accelerator provides an ecosystem of certified partners.

Appendix

Useful Links

Below are some useful documents to explore further:

Confluent Cloud Documentation:

- Overview of Confluent Cloud
- Kafka Cluster Types in Confluent Cloud
- Hybrid and Multicloud Architecture with Apache Kafka
- Kora | The Cloud Native Kafka Engine | Elastic, resilient & fast
- confluentinc/confluent | Terraform Registry
- Confluent Cloud Metrics
- Custom Connectors for Confluent Cloud

Additional Migration Resources:

- White Paper: Data at Rest Migrations
- Migration Accelerator Service
- Confluent Partner Migration Accelerator Program

Confluent Client Documentation:

- Build Kafka Client Applications on Confluent Cloud
- Optimize and Tune Confluent Cloud Clients
- Client Producer and Consumer Configuration Recommendations for Confluent Cloud
- Connect clients and applications to Confluent Cloud

Cluster Linking:

- Linking Self-Managed and Confluent Cloud Clusters for Hybrid Cloud Deployments
- Migrate Data with Cluster Linking on Confluent Cloud
- Supported Cluster Types | Cluster Linking
- Confluent Kafka Mirror Promote

Confluent Replicator:

- Overview of Confluent Replicator
- Replicator Configuration Reference
- Confluent Replicator to Confluent Cloud Configurations
- Use Schema Registry to Migrate Schemas
- Confluent for Kubernetes: Replicator Example
- Replicator and Cross-Cluster Failover

Schema Linking:

- Schema Linking Quick Start Tutorial
- Migrate Schemas to Confluent Cloud

Data Governance:

- Overview of Schema Registry
- Stream Quality Overview
- Data Portal
- Stream Lineage
- Data Contracts for Schema Registry

Confluent Cloud Security:

- Bring Your Own (Encryption) Key
- Confluent OAuth
  - Group mappings
- API Keys
- Service Accounts
- User Accounts
- ACLs
- RBAC

Confluent Cloud Networking:

- Use Public Egress IP Addresses on Confluent Cloud for Connectors and Cluster Linking
- Networking on Confluent Cloud
- Confluent Cloud Networking Basics and Apache Kafka Connectivity
- Support Article: How to Use a Customer Truststore if connecting to Confluent Cloud (available for Confluent customers only)

CONFLUENT

Solution Brief

CONFLUENT MIGRATION ACCELERATOR

Jumpstart Your Data Streaming Journey

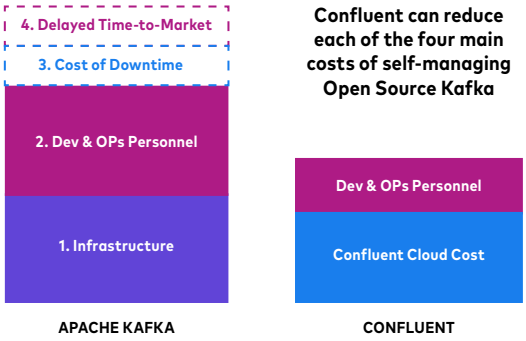
Migrate from traditional messaging systems or open-source Kafka deployments to Confluent.

Apache Kafka® is used by over 70% of the Fortune 500 to modernize their data architectures and power real-time applications that deliver rich, digital customer experiences and data-driven business operations.

It's become the central nervous system for modern enterprises, enabling them to connect, process, and react to all their data streams in real-time.

However, as enterprises adopted Apache Kafka® and other traditional messaging systems like TIBCO, IBM MQ, or ActiveMQ, they faced challenges in scale, performance, management, and security. These challenges limit innovation and increase the risk of system outages. Self-managing Kafka also creates significant ongoing operational burdens and risks.

Migrating to Confluent from traditional messaging systems or open-source Kafka deployments requires ample planning, skilled staff, engineering time, and upfront investments. This process can be daunting without expert guidance.



The Confluent Migration Accelerator aims to alleviate the challenges and complexities associated with migration, enabling organizations to focus on high-value projects instead of spending valuable time and resources on low-level tooling and maintaining platform performance. By migrating to the leading data streaming platform, you can save money while transitioning to a complete, cloud-native platform that can be deployed anywhere.

Key benefits of the Confluent Migration Accelerator include:

- **Save money and time -** Move to Confluent to unlock significant cost savings in comparison to traditional data streaming platforms. This has resulted in **40-60% of TCO savings**, in comparison to open-source Kafka, for our customers.
- **Press the migration easy button -** Partner with leading data streaming experts to augment staff, access funding credits (subject to approval), and accelerate the migration with packaged migration offerings. We have a global network of Confluent-certified system integrators who are ready to help assist the switch to Confluent.
- **Upgrade with confidence -** Migrate to the **leading cloud-native and complete data streaming platform** that can run everywhere and in any combination of on-prem, hybrid cloud, and multicloud deployments. Multi-tenancy, elasticity, data balancing, network costs, and more are all optimized with our cloud-native platform to reduce spend on infrastructure. Moving to Confluent will help automate every layer of the stack to monitor, debug, and build with the most complete data streaming platform on the market.

1M+ hours	60% lower	<6 months
of Kafka experience to help you migrate like a pro	TCO compared to open-sourced Kafka	to propel your streaming use cases in production

With **Confluent's Migration Accelerator**, your data streaming evolution just got faster. Leverage our partner ecosystem to augment your team, fund migrations (subject to approval), and accelerate your migration journey with tailored migration offerings.

Confluent Cloud Partners

Together with our Cloud Partners, we can help you take the first steps towards a successful cloud migration by providing funding credits (subject to approval).

aws Azure Google Cloud

Confluent System Integrator Partners

Leverage strategic collaborations with System Integrators (SIs) to ensure a seamless transition to a contemporary, event-driven architecture at your pace. Tap into a wealth of expertise and tailored partner solutions to effortlessly integrate our offerings into your infrastructure.

improving

Improving's migration workshop methodology covers everything from use case identification to roadmap development and important considerations for building robust event-driven architectures with Confluent.

iLink Digital

iLink Digital's migration strategy includes an automation-first approach, improving adoption and increasing consumption.

NESS

Ness' migration readiness assessment is a focused, one-week project accelerator preparing engineering organizations to plan, design, and implement streaming-based applications.

<epam>

EPAM utilizes migVisor Streaming Migration accelerator as part of the migVisor Cloud end-to-end migration enablement suite to enable their clients in completing event streaming migrations.

syncopate

Psyncopate's full day, no cost workshop provides proven migration capabilities from Legacy / Messaging Implementations.

SVA

SVA System Vertrieb Alexander GmbH's migration best practices include everything from hosting a Data Value workshop to prototyping, implementation, and deployment with long-term operation and support.

CloudMile

CloudMile's migration framework provides a comprehensive solution, covering assessment, implementation, and optimization to help customers expedite their digital transformation.

platformatory

Platformatory's migration approach helps customers migrate from Kafka-compatible platforms and other messaging systems with no risk of disruption.

Learn more about the Confluent Migration Accelerator.

Unpacking the migration process with Improving

Assess the challenges and benefits of a cloud migration and learn how Confluent's partner, Improving, can help you simplify and navigate the process with confidence.

In this webinar, Improving—a modern digital services company that provides enterprise software consulting, development, and training to Fortune 500 and Global 1000 enterprises—will showcase their Migration Workshop methodology, which covers everything from use case identification to roadmap development, as well as important considerations for building robust event-driven architectures with Confluent.

Learn how Improving can help you accelerate your migration journey to Confluent, allowing you to focus on day-to-day activities rather than managing low-level infrastructure.

How BigCommerce migrated to Confluent

BigCommerce—a leading e-commerce company that supports merchants, such as Solo Stove, Skullcandy, and Yeti—had an ETL batch-based system. Their merchants had to wait 8 hours before pulling any analytics and insights reporting (e.g., who is coming to their site, adding to their carts, popular products). The system consisted of 30 odd map-reduce jobs, which would sometimes fail and need manual intervention. This online talk showcases how BigCommerce transitioned their legacy ETL batch-based systems to an event-driven architecture in a four-phased approach.

Discover why BigCommerce, a leading e-commerce platform, decided to modernize their approach with a fully managed Kafka platform with the help of Google Cloud and Confluent. Explore how they migrated 1.6 billion events per day from Kafka to Confluent, saving 20+ hours a week in Kafka management in just 5 months—and learn how you can, too!

CONFLUENT

Case Study

WWW.CERVED.COM

Cerved Expedites Strategic Decision-Making with Real-Time Data Access from Confluent

Cerved

Headquarters
Milan, Italy

Industry
Financial Services

Challenge
Distributed databases using different technologies were causing slow query times and high latency as the system struggled to locate and process data at speed—which negatively impacted service delivery.

Solution
Cerved partnered with Confluent to provide instant access to business-critical data to empower strategic decisions and support.

Results
- Near-real time data propagation (event-driven architecture)
- Increased availability with Confluent's managed service that's helped build resilient systems with a 99.99% SLA
- Increased security with automatic monitoring and alarming
- Increased scalability with a higher number of projects and automated expansion of platform capacity
- Cost savings thanks to reduced infrastructure costs and governance, with 24/7 dedicated support from Confluent Cloud
- Disaster recovery that's automatically updated with Cluster Linking

Technical Solution
- Advanced Stream Governance
- Schema Registry
- Connectors
  - Amazon S3 Sink Connector
  - Databricks Delta Lake Sink Connector

Cerved is an IT company and leading provider of credit information, market advice, and risk intelligence serving a variety of sectors. Over 30,000 businesses and 95% of Italian banks use Cerved to identify new strategies, protect themselves from risk, improve performance, and grow sustainably.

Data is the driving force behind Cerved's continued success. But when that data is split across multiple platforms that don't communicate, accessing key insights is time-consuming and wastes resources that could be better used elsewhere.

To provide a faster and more effective B2B service, Cerved needed a streaming platform that could handle huge amounts of data and form the basis for its new, connected ecosystem.

In 2017, Confluent provided Cerved with an on-premises platform to manage faster access to business-critical data at the touch of a button. The information services provided could now deliver strategic solutions and risk management support to the business in near real time.

Solving the primary challenge: difficulty decoding data

Cerved was looking to build a unique data ecosystem that could access public, Chamber of Commerce, web, and social data—alongside data from business partners like Experian and Assilea. These datasets provide their customers with critical information and insights to help them improve decision-making and gain a competitive advantage over their competitors.

"To give our customers the most relevant, up-to-date, and reliable advice, we need continual access to data. But as all of our data runs through different technologies, and we've got lots of applications communicating with each other simultaneously, it was a big issue to manage the entire infrastructure."

— PASQUALE FOSSO, HEAD OF DATA ARCHITECTURE AT CERVED

But running one of the biggest data ecosystems in Italy isn't easy. Cerved has hundreds of databases, with millions of potential data points that teams need to locate, access, and transmit. Data has to be available on demand to scan, capture, identify, and interrogate every piece of information needed to paint a clear picture of financial risks and market opportunities.

Event-driven architecture

In a big push for digitization across the company, Cerved needed a better way to facilitate data access between its disparate databases and systems.

"Managing an event-driven architecture is always complex and can be pretty debilitating when it impacts service delivery. Thankfully, with Confluent, we have found the best solution for our event-driven approach even when accessing a large volume of data, which helps us deliver insights much faster," said Fosso.

Cerved quickly realized many benefits using Confluent Platform, but managing on-premises clusters internally was too resource intensive. To accelerate new strategic projects and reduce operational costs, the company adopted Confluent Cloud.

The migration to Confluent Cloud

The migration project started in the latter part of 2022, with the entire production environment deployed within six months. The process was managed by Cerved's Data Platform team, which involved 12 dev teams across the Italian territory.

Overall, 200 applications, 789 topics, and 2,452 partitions with 1,107 schemas were migrated. In total there are now 40,000 connected clients with 2TB of storage.

The applications are based on languages such as Scala, Python, Java, and .NET with development/connection frameworks including Spring Boot, Kafka Streams, Spark, and Apache Kafka® connectors.

"Availability of data is crucial, and having a fully managed platform for the data distribution lets us provide solutions to our customers at speed," explained Fosso.

Enjoying the results of migrating to Confluent Cloud

With a unique and consolidated data ecosystem using Confluent Cloud, Cerved has seen some great results, including:

Around-the-clock availability

With Confluent's SaaS, Cerved has a network of resilient systems that are always on and continually adhere to the 99.99% SLA.

Strengthened security

Cerved uses Confluent Cloud to transfer large volumes of data in near real time and in a secure platform that implements encryption at rest on both the stored and in transit data for all the consumers of the platform.

Increased scalability

Thanks to the flexibility of Confluent's managed service, Cerved is able to increase the number of new projects, without having to worry about infrastructure sizing, because the managed platform automatically adjusts any necessary expansions of the clusters.

With Confluent, this process is quick and simple, letting teams focus on what they do best: providing strategic B2B services to their customers.

Cost-savings thanks to fully managed support

With a fully managed Kafka service from Confluent Cloud, Cerved has entirely offloaded the operational burden which helps them reduce infrastructure and management expenses.

"With Confluent at the helm, we've cut back significantly on operating expenses, as we no longer need to waste time chasing data, managing capacity, or maintaining our growing Kafka deployment," said Fosso.

Disaster recovery

Implementing a disaster recovery strategy with Cluster Linking increases the availability and reliability of mission-critical applications, helping to minimize data loss and downtime in the event of an unexpected disaster.

What's next for Cerved?

Cerved can now access critical information across its ecosystem in a matter of seconds, which helps its growing team of consultants turn raw data into strategic insights and innovative ideas faster than ever before.

But what does the future hold for the business intelligence giant?

"In the coming months, we're looking into the possibility of broadening our use cases of Confluent, as it's become one of the most important pillars of our data mesh architecture," said Fosso.

Learn More About Cerved: https://www.cerved.com/