Cloudera Runtime 7.1.7

Streams Replication Manager Overview

Date published: 2019-09-13
Date modified: 2021-08-05

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Overview 4

Key Features 5

Main Use Cases 6

Use Case Architectures 7
- Highly Available Kafka Architectures 7
  - Active / Stand-by Architecture 7
  - Active / Active Architecture 8
  - Cross Data Center Replication 9
- Cluster Migration Architectures 10
  - On-premise to Cloud and Kafka Version Upgrade 10
  - Aggregation for Analytics 12

Streams Replication Manager Architecture 12
- Streams Replication Manager Driver 12
  - Connect workers 13
  - Connectors 15
  - Task architecture and load-balancing 17
  - Driver inter-node coordination 19
- Streams Replication Manager Service 19

Understanding Replication Flows 20
- Replication Flows Overview 20
- Remote Topics 21
- Bidirectional Replication Flows 22
- Fan-in and Fan-out Replication Flows 22

Understanding co-located and external clusters 23

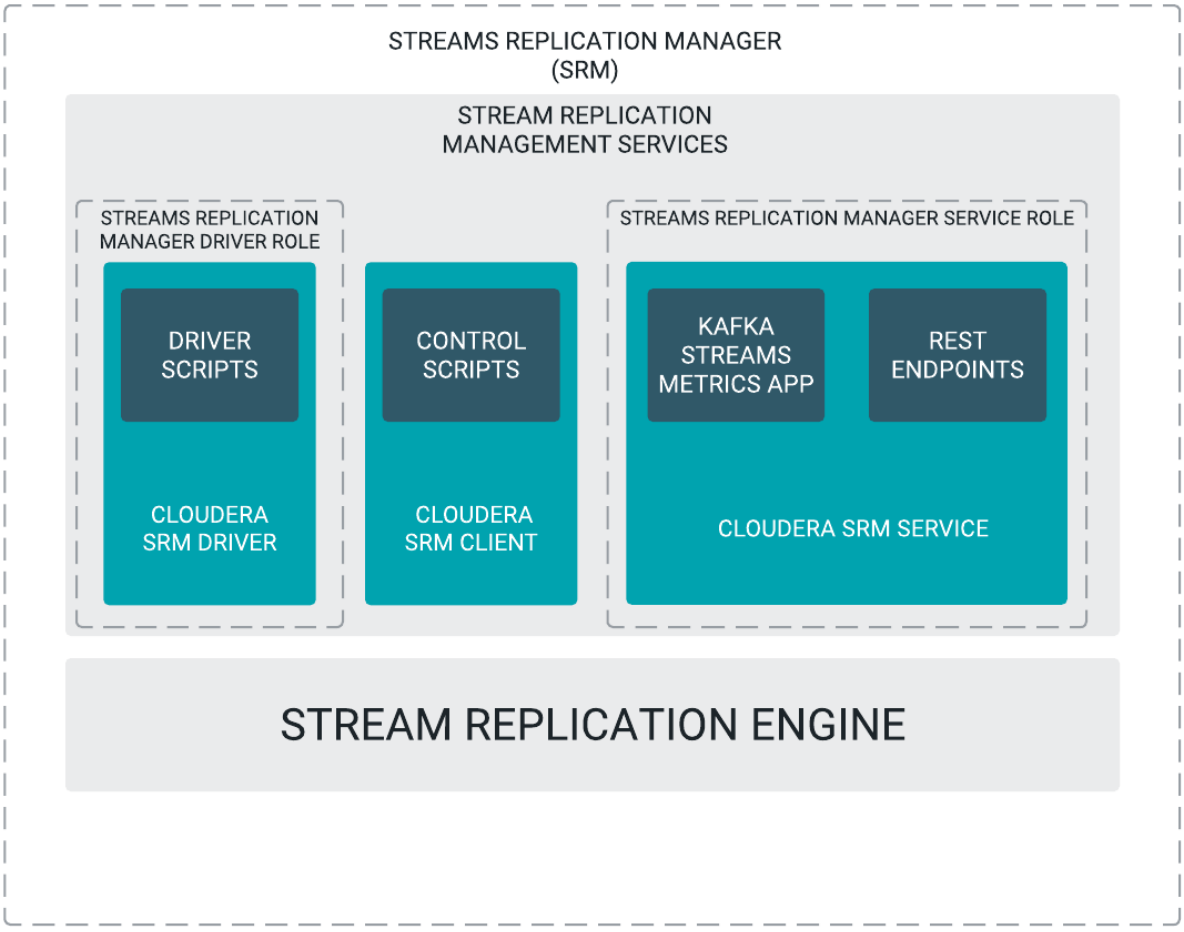
Understanding SRM properties, their configuration and hierarchy 27

Overview

Get familiar with Streams Replication Manager and its components.

Streams Replication Manager (SRM) is an enterprise-grade replication solution that enables fault tolerant, scalable and robust cross-cluster Kafka topic replication. SRM provides the ability to dynamically change configurations and keeps the topic properties in sync across clusters at high performance. SRM also delivers custom extensions that facilitate installation, management and monitoring making SRM a complete replication solution that is built for mission critical workloads. Streams Replication Manager consists of two main components. The Stream Replication Engine and the Stream Replication Management Services.

Figure 1: Streams Replication Manager Overview



Stream Replication Engine

The Stream Replication Engine is a next generation multi-cluster and cross-datacenter replication engine for Kafka clusters.

Stream Replication Management Services

Stream Replication Management Services are services powered by open source Cloudera extensions which utilize the capabilities of the Stream Replication Engine. These services provide:

- Easy installation
- Lifecycle management

- Management and monitoring of replication flows across clusters

The Stream Replication Management Services includes the following custom extensions:

Cloudera SRM Driver

The Cloudera SRM Driver is a small wrapper around the Stream Replication Engine that adds the extensions provided by Cloudera. It provides the ability to spin up SRM clusters and has a metrics reporter. The driver is managed by Cloudera Manager and is represented by the Streams Replication Manager Driver role.

Cloudera SRM Client

The Cloudera SRM Client provides users with command line tools that enable replication management for topics and consumer groups. The command line tool associated with the Cloudera SRM Client is called srm-control.

Cloudera SRM Service

The Cloudera SRM Service consist of a REST API and a Kafka Streams application to aggregate and expose cluster, topic and consumer group metrics. The service is managed by Cloudera Manager and is represented by the Streams Replication Manager Service role.

Key Features

SRM has the following main features.

Remote topics

Remote topics are replicated topics referencing a source cluster via a naming convention. For example, the “topic1” topic from the “us-west” source cluster creates the “us-west.topic1” remote topic on the target cluster. SRM automatically applies this configurable "replication policy" across your organization, enabling tooling to distinguish remote topics from source topics. For more information regarding remote topics, see Understanding Replication Flows.

Consistent semantics

Partitioning and record offsets are synchronized between replicated clusters to ensure consumers can migrate from one cluster to another without losing data or skipping records.

Cross cluster configuration

Topic-level configuration properties are synced across clusters. For example, the cleanup policy (cleanup.policy), or the log segment file size (segment.bytes), as well as other topic-level configurations are automatically synched to remote topics. This simplifies managing topics across multiple Kafka clusters.

Consumer group checkpoints

In addition to data and configuration, SRM replicates consumer group progress via periodic checkpoints. At configurable intervals, checkpoint records are emitted to downstream clusters, encoding the latest offsets for whitelisted consumer groups and topic-partitions. As with topics, groups are matched against an allowlist which can be updated dynamically with srm-control. Normally, consumer group offsets are not portable between Kafka clusters, as offsets are not consistent between otherwise identical topic-partitions on different clusters. SRM’s checkpoint records account for this by including offsets which are automatically translated from one cluster to another. This offset translation feature works in both directions; a consumer group can be migrated from one cluster to another (failover) and then back again (failback) without skipping records or losing progress.

Automatic topic and partition detection

SRM monitors Kafka clusters for new topics, partitions, and consumer groups as they are created. These are compared with configurable whitelists, which may include regular expressions.

Tooling to automate consumer migration

SRM tooling enables operators to translate offsets between clusters and to migrate consumer groups while preserving state.

Centralized configuration for multi-cluster environments

SRM leverages a single top-level configuration file to enable replication across multiple Kafka clusters. Moreover, command-line tooling can alter which topics and consumer groups are replicated in real-time.

Replication monitoring

Since cluster replication will mainly be used for highly critical Kafka applications, it is crucial for customers to be able to easily and reliably monitor the Kafka cluster replications. The custom extensions included with SRM collect and aggregate Kafka replication metrics and make them available through a REST API. This REST API is used by Streams Messaging Manager (SMM) to display metrics. Customers could also use the REST API to implement their own monitoring solution or plug it into third party solutions. The metrics make the state of cluster replication visible to end users who then can take corrective action if needed.

Custom replication policies

The replication policy used by SRM defines the basic rules of how SRM replicates data. While the default replication policy shipped with SRM is the Cloudera recommended and supported replication policy, custom developed replication policies can be used. Developing and using your own replication policy enables you to gain full control over how SRM replicates data.

Important: Cloudera provides limited support for deployments that use a custom replication policies. Additionally, understand that some key features including replication monitoring with the SRM Service will not work if a custom replication policy is in use.

Related Information

Monitoring Cluster Replications Overview

Understanding Replication Flows

Main Use Cases

Learn about the main use cases of SRM.

Apache Kafka has become an essential component of enterprise data pipelines and is used for tracking clickstream event data, collecting logs, gathering metrics, and being the enterprise data bus in a microservices based architectures. Kafka supports internal replication to support data availability within a cluster. However with Kafka based applications becoming critical, enterprises require that the data availability and durability guarantees span entire cluster and site failures.

Replication of data across clusters and sites is key for the following use cases:

Disaster Recovery

Common enterprise use cases for cross-cluster replication is for guaranteeing business continuity in the presence of cluster or data center-wide outages.

Aggregation for Analytics

Aggregate data from multiple streaming pipelines possibly across multiple data centers to run batch analytics jobs that provide a holistic view across the enterprise.

Data Deployment after Analytics

This is the opposite of the aggregation use case in which the data generated by the analytics application in one cluster (say the aggregate cluster) is broadcast to multiple clusters possibly across data centers for end user consumption.

Isolation

Due to performance or security reasons, data needs to be replicated between different environments to isolate access. In many deployments the ingestion cluster is isolated from the consumption clusters.

Geo Proximity

In geographically distributed access patterns where low latency is required, replication is used to move data closer to the access location.

Cloud Migration

As more enterprises have an on-premise and cloud presence, Kafka replication can be used to migrate data to the public or private cloud and back.

Legal and Compliance

Much like the isolation uses case, a policy driven replication is used to limit what data is accessible in a cluster to meet legal and compliance requirements.

Use Case Architectures

Highly available and cluster migration architecture examples for SRM.

Highly Available Kafka Architectures

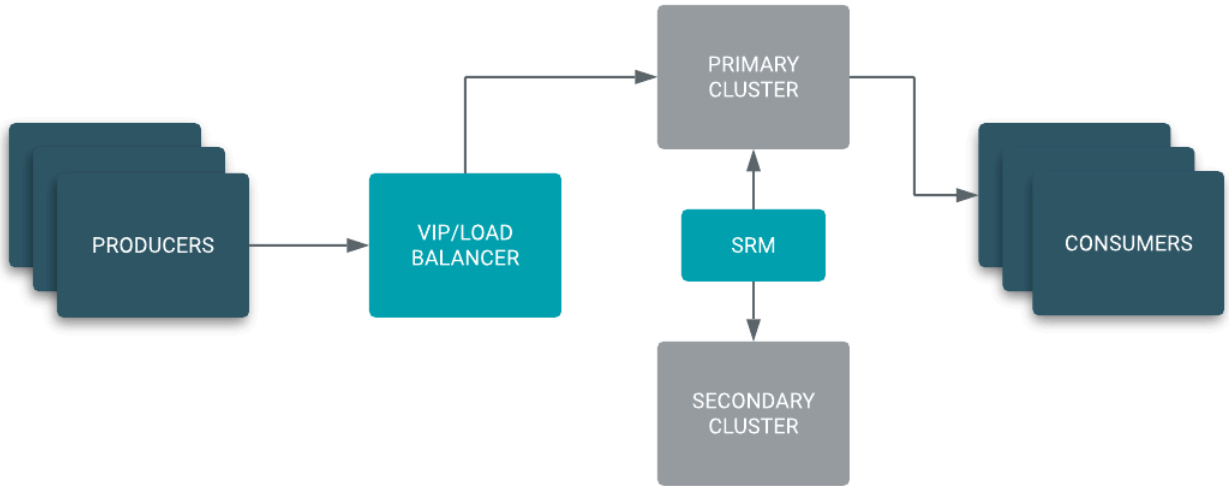
A highly available Kafka deployment must be able to survive a full single cluster outage while continuing to process events without data loss. With SRM, you can implement highly available Apache Kafka deployments which either follow an Active / Stand-by or an Active / Active model.

Active / Stand-by Architecture

Active / Stand-by architecture example for SRM.

In an Active / Stand-by scenario, you set up two Kafka clusters and configure SRM to replicate topics bidirectionally between both clusters. A VIP or load balancer directs your producers to ingest messages into the active cluster from which consumer groups are reading from.

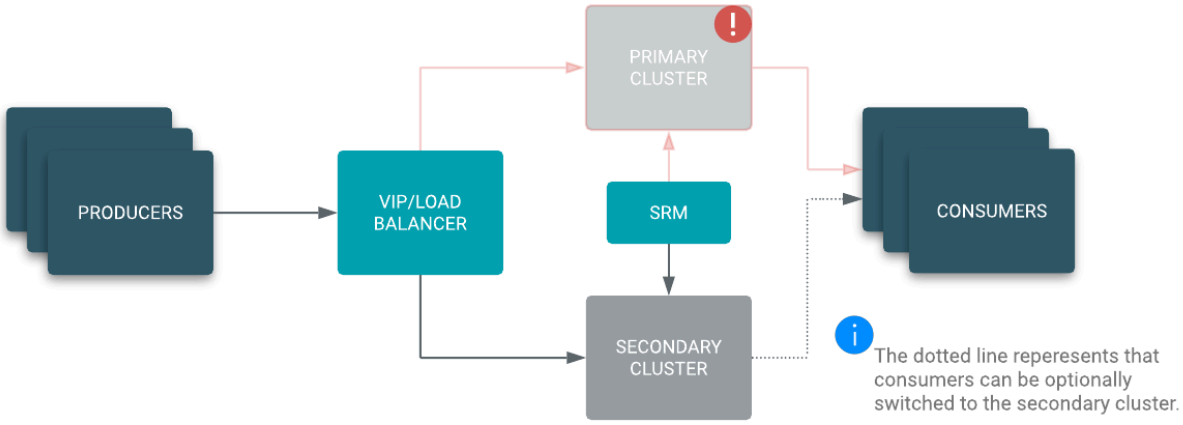
Figure 2: Active / Stand-by Architecture Standard Operation



In case of a disaster, the VIP or load balancer directs the producers to the stand-by cluster. You can easily migrate your consumer groups to start reading from the stand-by cluster or simply wait until the primary cluster is restored if the resulting consumer lag is acceptable for your use case.

While the primary cluster is down, your producers are still able to ingest. Once the primary cluster is restored, SRM automatically takes care of synchronizing both clusters making failback seamless.

Figure 3: Active / Stand-by Architecture Cluster Failure



Implementing an Active/Stand-by architecture is the logical choice when an existing disaster recovery site with established policies is already available, and your goals include not losing ingest capabilities during a disaster and having a backup in your disaster recovery site.

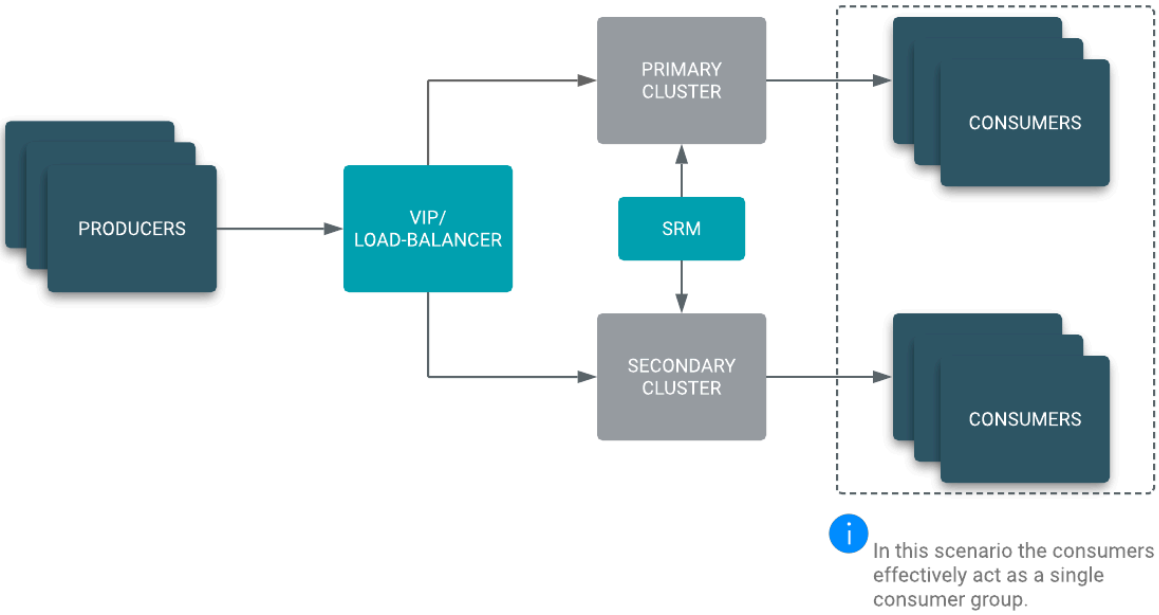
Active / Active Architecture

Active / Active architecture example for SRM.

In an Active / Active scenario, your producers can be load balanced to either your primary or secondary cluster. SRM is configured to replicate topics bidirectionally between both clusters. What makes this architecture Active / Active, is the fact that you now have consumers reading from both clusters at the same time, essentially acting like a cross-cluster consumer group. In case of a disaster the VIP or load balancer directs the producers to the secondary cluster and the secondary cluster consumer group is still able to process messages.While the primary cluster is down, your

producers are still able to ingest and your consumers are still able to process messages. This results in a 0 downtime and hands-off failover in case of a disaster. Once the primary cluster is back online, SRM automatically takes care of synchronizing both clusters and your primary consumer group resumes processing messages.

Figure 4: Active / Active Architecture



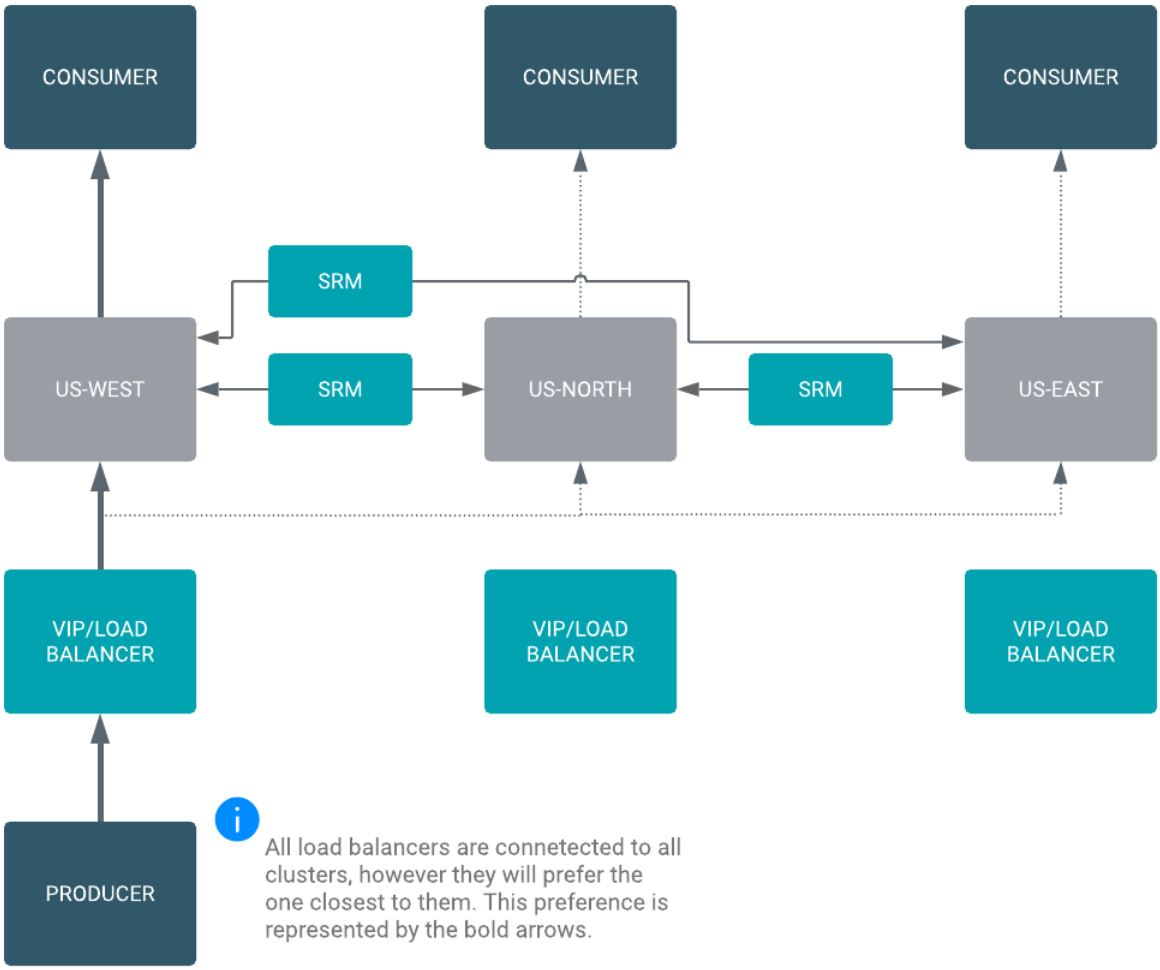
Cross Data Center Replication

Cross data center replication architecture example for SRM.

Certain applications not only require local high availability within one data center or one availability zone, but have to be highly available across data centers too. You can use SRM to set up replication between Kafka clusters in different data centers which results in messages being available to consumers in each of your data centers.

A load balancer directs your producers to the local data center or closest data center if the primary data center is down. SRM is configured to replicate topics between all data centers. If you are using more than two data centers, SRM is configured to create a “replication circle”, ensuring a single data center failure (for example us-north in the example below) does not halt replication between the remaining clusters.

Figure 5: Cross Data Center Replication Architecture



Cluster Migration Architectures

Example cluster migration architectures.

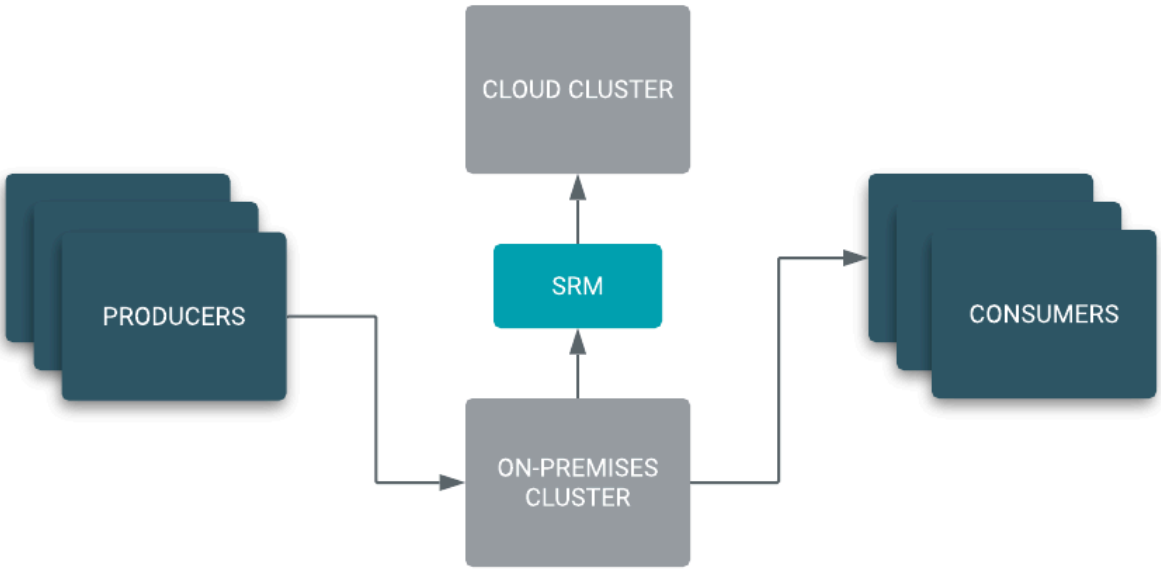
On-premise to Cloud and Kafka Version Upgrade

On-premise to cloud and Kafka version upgrade example architectures for SRM.

If you have an on-premises Apache Kafka cluster that you want to migrate to the cloud, not only do you have to migrate consumers and producers, you also have to migrate topics and their messages to the new cloud based cluster.

After you have set up replication through SRM, you only need to point your consumers to the new brokers before you can start processing messages from the cloud cluster. This approach ensures that the historical data kept in the on-premises Kafka cluster is migrated to the cloud cluster allowing you to replay messages directly from the cloud without having to go back to your on-premises cluster.

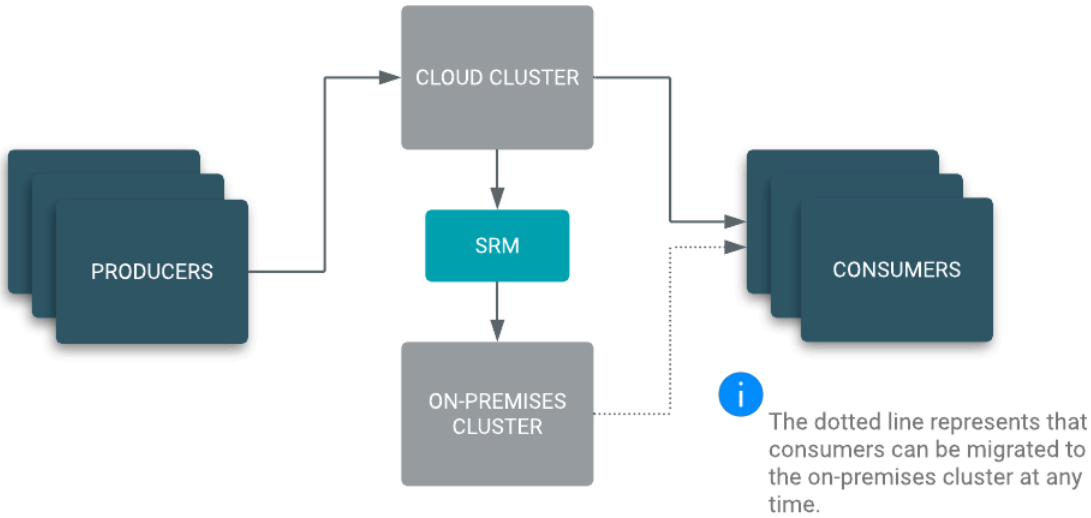
Figure 6: Cluster Migration On-premise



Producers and Consumers are using the on-premises cluster while SRM is replicating messages.

Once you have migrated your cluster, producers, and consumers to the cloud, you can use SRM to turn-around the replication direction and use the on-premises cluster as your DR cluster.

Figure 7: Cluster Migration Cloud



Producers and Consumers have been migrated to the cloud cluster and the on-premises cluster is used for disaster recovery.

Kafka Version Upgrade

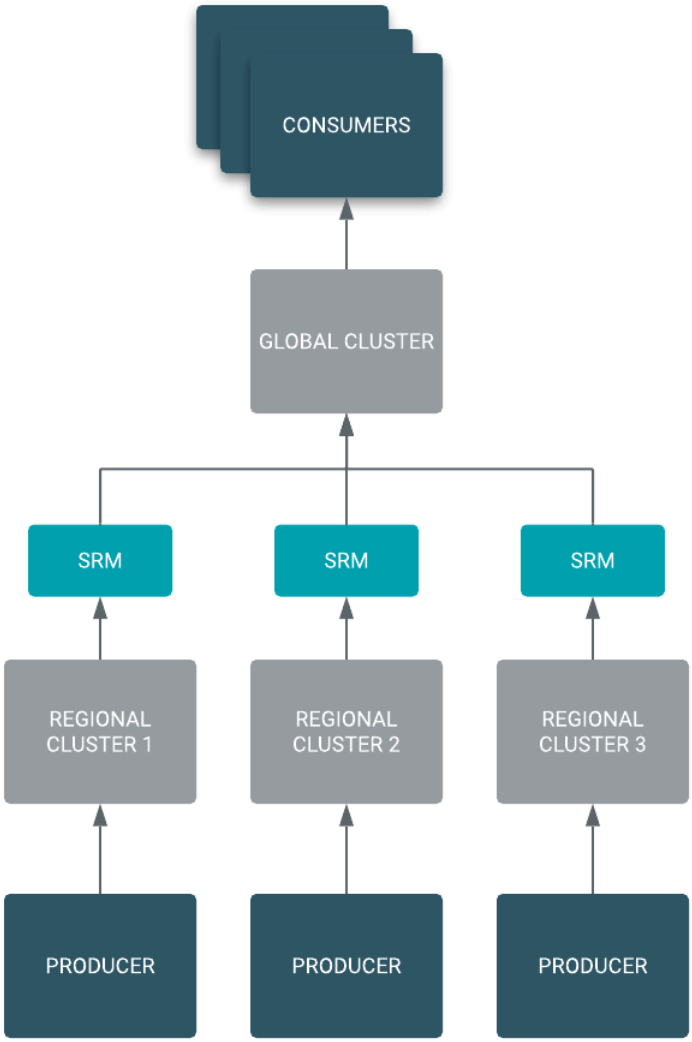
If you have to upgrade your Kafka cluster to a newer version and an in-place upgrade is not possible, you can use the same migration approach to provision a new cluster, use SRM to replicate all existing topics and messages before migrating your producers and consumers to interact with the new cluster.

Aggregation for Analytics

Aggregation for analytics architecture example for SRM.

SRM can be used to aggregate data from multiple streaming pipelines, possibly across multiple data centers, to run batch analytics jobs that provide a holistic view across the enterprise.

Figure 8: Aggregation for Analytics



Streams Replication Manager Architecture

Learn about the architecture of the Streams Replication Manager Driver and Service, which are two main components (roles) that make up Streams Replication Manager

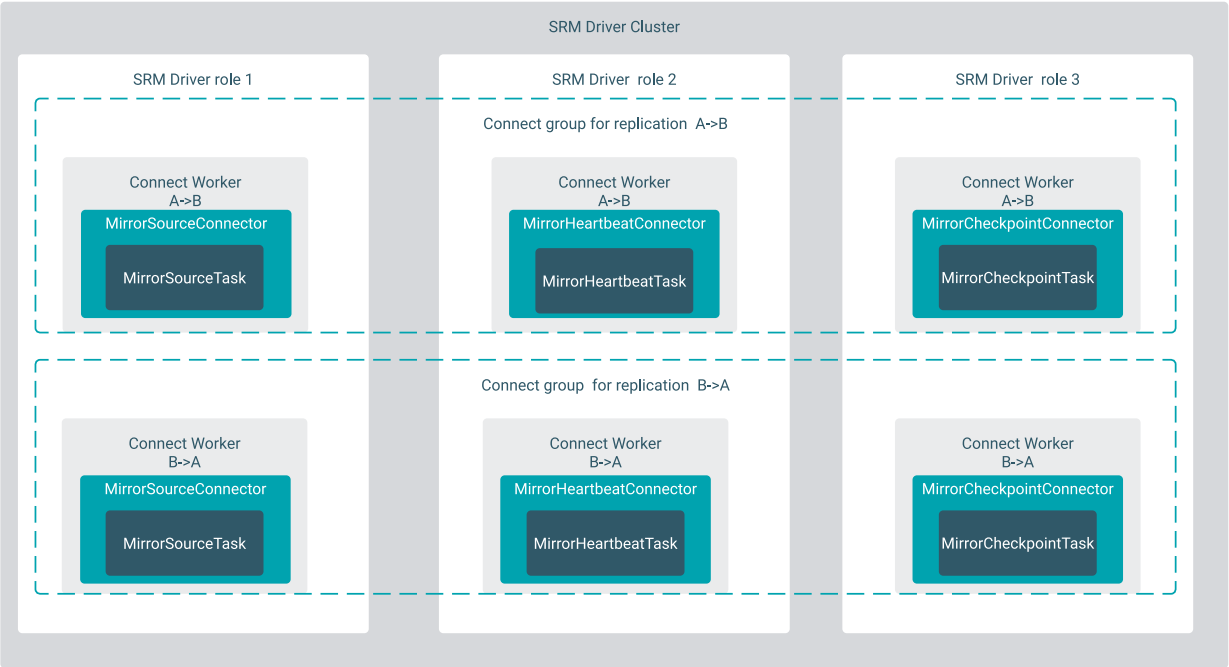
Streams Replication Manager Driver

The Streams Replication Manager Driver is responsible for executing the configured replication work between Kafka clusters. This includes data replication, consumer group offset replication, and heartbeating.

The Streams Replication Manager Driver role (SRM Driver) is built on top of the Kafka Connect framework and utilizes a group of connectors to execute replication. While Kafka Connect is bound to a single Kafka cluster by design, the SRM Driver must connect to multiple Kafka clusters. The SRM Driver achieves this by wrapping multiple Connect workers in a single driver process. Specifically, for each possible replication between the configured clusters, the SRM Driver spins up a separate Connect worker. The Connect workers join a Connect group which is dedicated to a single replication in the target Kafka cluster. Afterwards, each Connect group creates a single MirrorSourceConnector, a single MirrorCheckpointConnector and a single MirrorHeartbeatConnector. These Connectors and the task instances they generate are responsible for different aspects of the replication work.

When SRM Driver High Availability is in use, the Connect workers that are tied to a specific replication, but are running in different SRM Driver processes, coordinate through the Connect group protocol and balance the load.

Note: The following diagram is a visual representation of the Driver's architecture and showcases the different components of the Driver. In a running Driver, a single Connect worker can have a single, multiple, or even no Connectors or tasks assigned to it. As a result of this, the Connector and task distribution shown in this diagram does not fully reflect how Connectors and tasks are distributed in a running Driver. For more information on how Connectors and tasks are distributed, see Task architecture and load-balancing.

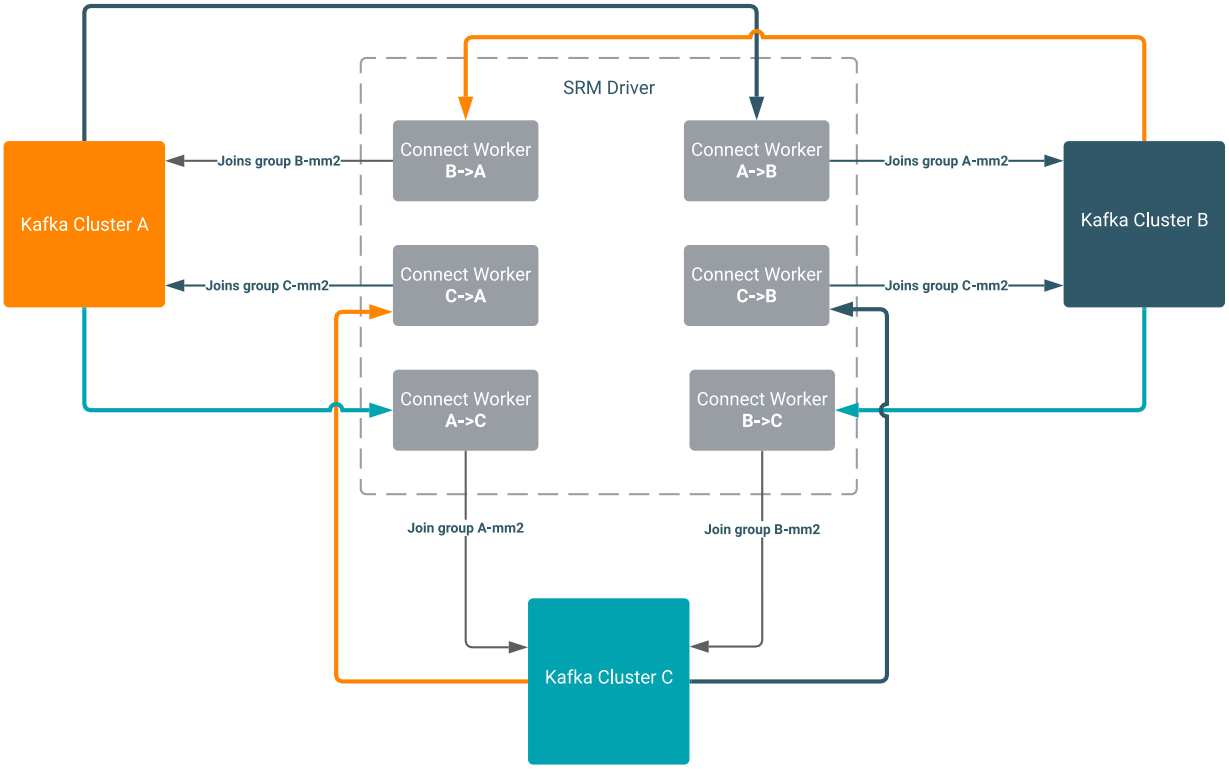


Connect workers

Learn about the Connect workers created by the Streams Replication Manager Driver.

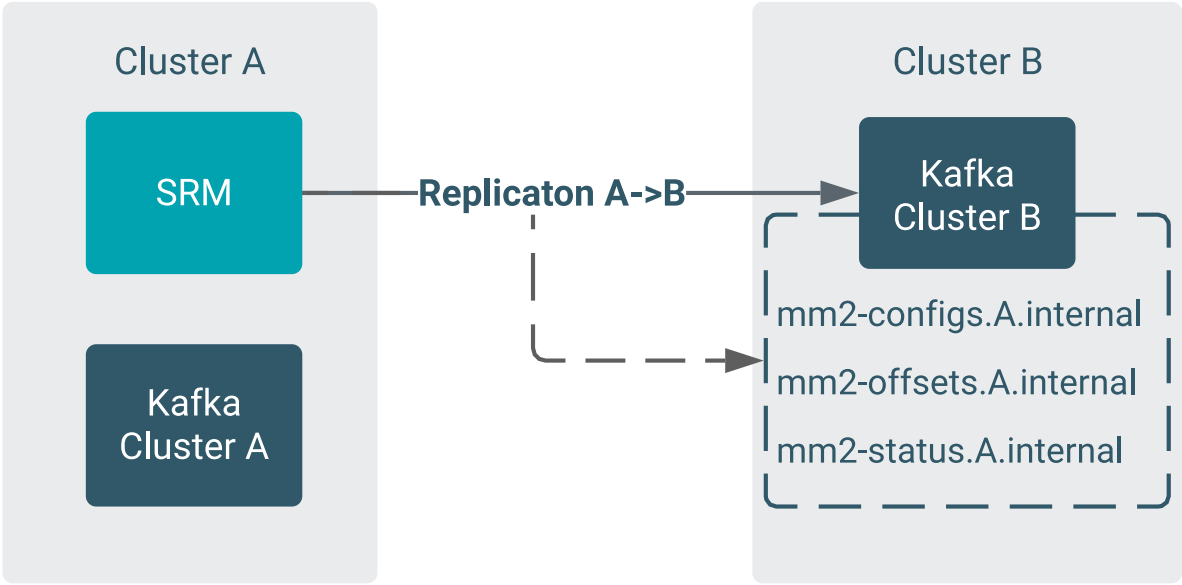
The Streams Replication Manager Driver role (SRM Driver) wraps multiple Connect workers in its process. Each Connect worker corresponds to a possible replication flow. At startup, if a target is specified for the replication in the Streams Replication Manager Driver Target Cluster property, a Connect worker is created for each possible cluster pair based on the aliases present in Streams Replication Manager Cluster alias. This means that for each possible replication, there is a running Connect worker, regardless of whether the replication is enabled. For enabled replications, the Connect worker creates and manages all three Connectors (MirrorSourceConnector, MirrorCheckpointConnector, and MirrorHeartbeatConnector). For disabled replication, the Connect worker only creates and manages a MirrorHeartbeatConnector. The MirrorHeartbeatConnector is spun up to ensure that the heartbeats topic is created on all clusters which might be the source of a replication flow.

The Connect workers always coordinate using the target Kafka cluster. They join a Connect group which is dedicated to a specific replication. This means that even when there are multiple replications targeting the same cluster, the replications are managed and load-balanced separately, through dedicated Connect groups.



Connect internal topics

SRM creates a separate Connect cluster as well as three internal Kafka topics for each replication. The internal topics are used by the Connect clusters to store their state. These internal topics are all located in the target cluster of the replication. The topic names reference the source cluster alias.



The three internal topics are as follows:

- mm2-configs.*[***SOURCE ALIAS***]*.internal

  Stores the Connector and Task configurations. Expected to be a single partition topic with cleanup.policy=compact. The records of the topic are generated based on SRM's configuration at startup. Losing the data does not cause issues for SRM after the service is restarted.
- mm2-offsets.*[***SOURCE ALIAS***]*.internal

  Stores the committed source offsets of SRM. SRM uses this internal topic to track its progress in the replication of the source topic. Expected to be a multi-partition topic with cleanup.policy=compact. The records of the topic are crucial for tracking the state of replication. Losing the data causes SRM to restart the replication of source topics, which leads to data duplication in the target cluster.
- mm2-status.*[***SOURCE ALIAS***]*.internal

  Stores the current status of Connectors and Tasks. Expected to be a multi-partition topic with cleanup.policy=compact. The records of the topic are created for monitoring purposes and do not affect replication. Losing the data does not cause issues for SRM after the service is restarted.

All three internal topics are created by SRM at startup with the expected configurations. Cloudera does not recommend reconfiguring or deleting these topics manually. Doing so can cause issues with replication, which might result in data loss. However, if the topics are created with an incorrect configuration, manual reconfiguration is required. In a case like this, SRM must be stopped, and the topic properties must be updated with correct values. Updating topic properties can be done with the kafka-configs tool.

Related Information

kafka-configs

Connectors

Learn about the different Connector implementations created by the Connect workers of the Streams Replication Manager Driver.

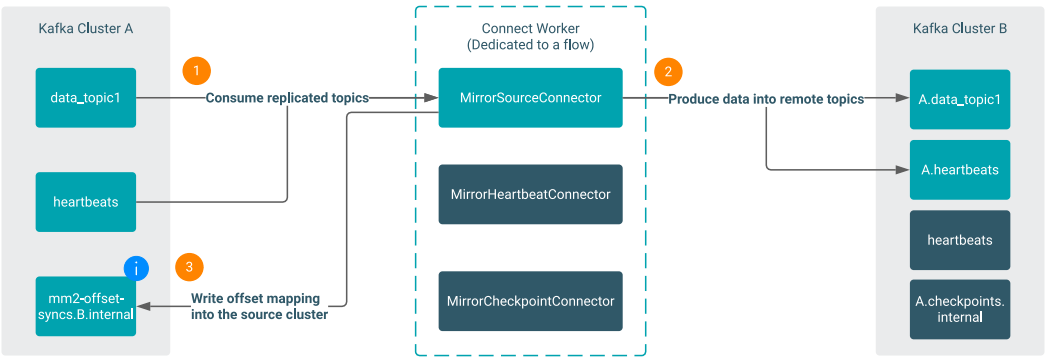
The Streams Replication Manager Driver role (SRM Driver) employs multiple Connect workers to execute replication, one per each replication. Inside the Connect workers, three separate Connector implementations are utilized. These are the following:

- MirrorSourceConnector
- MirrorHeartbeatConnector
- MirrorCheckpointConnector

MirrorSourceConnector

The MirrorSourceConnector is responsible for replicating topics between the source and the target cluster. The topics to be replicated are defined by allow and deny lists, which can be manipulated using the srm-control tool.

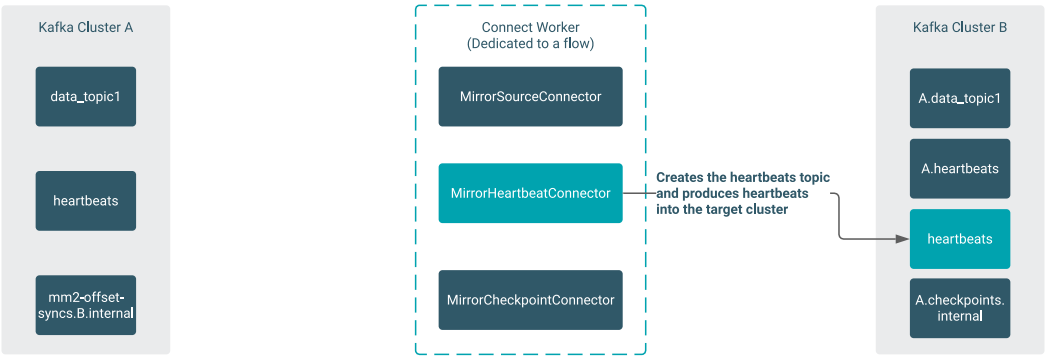
In addition to replicating data, the MirrorSourceConnector also manages the offset sync topic in the source cluster. When production into the target cluster is successful, the mapping between the source offset and the target offset is written into an offset sync topic in the source cluster. The offset sync topic is used by the MirrorCheckpointConnector.



MirrorHeartbeatConnector

The MirrorHeartbeatConnector is responsible for creating the heartbeats topic in the target cluster. It also periodically produces heartbeats into the heartbeats topic. The purpose of the heartbeats topic is twofold:

- It ensures that a topic is available at all times on the source clusters. This way, a replication always has at least a single topic that it can pick up. This functions as a reliable smoke test for the replication.
- It is used by the SRM Service to discover configured replications.

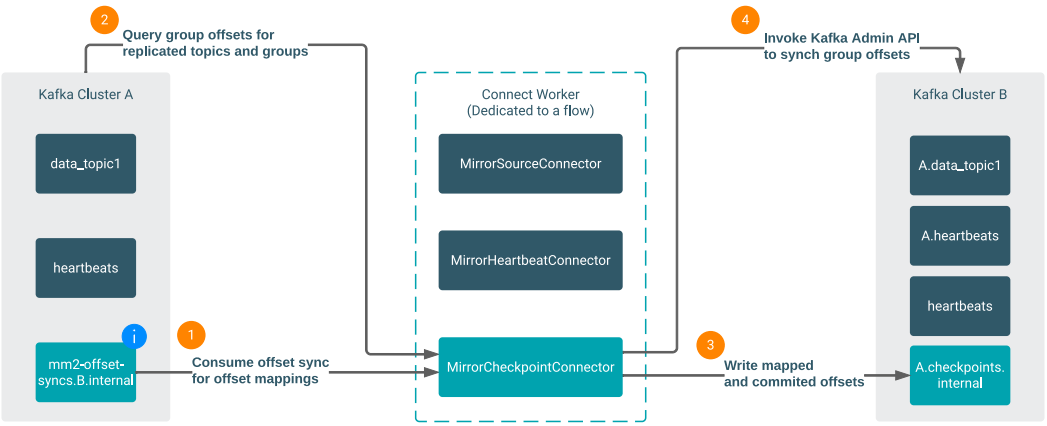


MirrorCheckpointConnector

The MirrorCheckpointConnector is responsible for replicating the committed group offsets between the source and target clusters. The offsets are calculated based on the offset sync topic managed by the MirrorSourceConnector. The offsets of the groups and the topic partitions that are replicated are defined by allow and deny lists, which can be manipulated using the srm-control tool.

The offsets are written into the checkpoint topic in the target cluster. The contents of the checkpoint topic can be exported with srm-control. Exported offsets can then be applied to the consumer groups in the target cluster with the kafka-consumer-groups Kafka tool.

In addition to writing the mapped offsets into the checkpoint topic, the connector is also capable of periodically applying the offsets to the consumer groups in the target cluster. This is done using the Kafka Admin API.



Task architecture and load-balancing

Learn how tasks are distributed and how load is balanced by the Streams Replication Manager Drivers that are running in the same cluster.

Streams Replication Manager Driver roles (SRM Driver) of the same cluster share the load of the replications among each other. They utilize the task load balancing of the Connect framework. Each Connector in a replication creates a set of task configurations. The number of tasks depend on the replicated topic partitions, groups, and the Tasks Max configuration of SRM. These task instances provide a way to parallelize the work in an SRM replication. After the Connectors create the task configurations, the Connect framework distributes the tasks between the SRM Drivers. More specifically, between the Connect workers running in the SRM Drivers corresponding to a specific replication flow. The task instances are as follows:

MirrorSourceTask

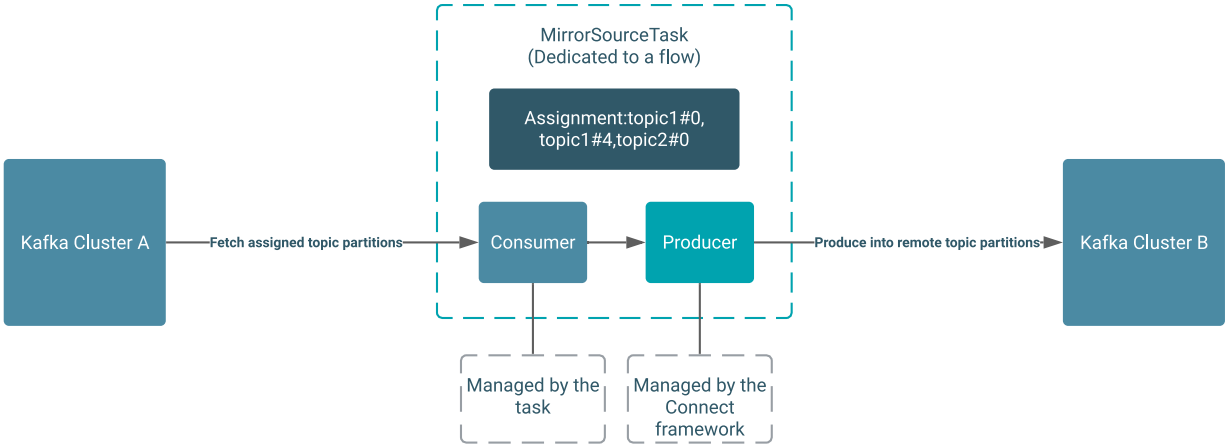
The MirrorSourceTask is created by the MirrorSourceConnector. It is responsible for executing data replication. A MirrorSourceTask wraps a consumer and a producer instance. The consumer instance is managed by the task. The producer instance is managed by the Connect framework.

Each task receives its assignment from the MirrorSourceConnector as a list of topic partitions. These are assigned to the consumer. The fetched records are then forwarded to the producer. The target topic name is generated based on the ReplicationPolicy.

The number of MirrorSourceTasks created in a replication is based on the following formula:

```
min(tasks.max, source_topic_partition_count)
```

In this formula, tasks.max corresponds to what Tasks Max is set to in SRM's configuration. The source_topic_partition_count is the number of replicated topic partitions, which is based on the topic allow and deny lists configured with the srm-control tool.



MirrorCheckpointTask

The MirrorCheckpointTask is created by the MirrorCheckpointConnector. It responsible for executing the consumer group offset sync. A MirrorCheckpointTask wraps an admin client and a producer instance. The admin client is managed by the task. The producer instance is managed by the Connect framework.

Each task receives its assignment from the MirrorCheckpointConnector as a list of consumer groups. The assigned consumer group offsets are periodically queried for the replicated topic partitions through the admin client, and are written into the target cluster's checkpoints topic. The replicated offsets can also be applied to the consumer groups in the target cluster by setting sync.group.offsets.enabled to true.

The number of MirrorCheckpointTasks created in a replication flow is based on the following formula:

```
min(tasks.max, source_consumer_group_count)
```

In this formula, tasks.max corresponds to what Tasks Max is set to in SRM's configuration. The source_consumer_group_count is the number of replicated consumer groups, which is based on the group allow and deny lists configured with the srm-control tool.

MirrorHeartbeatTask

The MirrorHeartbeatTask is created by the MirrorHeartbeatConnector. It is responsible for producing heartbeats into the target cluster's heartbeats topic. A MirrorHeartbeatTask wraps a producer instance that is managed by the Connect framework. In each replication, there is a single MirrorHeartbeatTask instance.

Task load balancing

Multiple Connect workers participate in each replication. This typically means that each SRM Driver of the cluster has one Connect worker dedicated to a specific replication.

After the Connect group is formed by the Connect workers, the Connectors are instantiated and task configurations are generated. This results in three Connectors, a number of MirrorSourceTasks and MirrorCheckpointTasks, as well as a MirrorHeartbeatTask. For a newly formed group, these Connectors and task instances are assigned to the Connect workers in a round robin fashion. When the group already has an existing assignment, and there is membership change (for example, a Connect worker joins or leaves the group), the Connectors and tasks are assigned in a cooperative and incremental manner. This allows for the majority of the work to continue without interruption.

Unless there are no changes made to the task configurations, meaning that no new topics and groups are added to the replication, the task definitions stay the same. However, based on Connect group membership changes, the tasks can be moved between workers. With classic Connect, these tasks can be managed separately through an admin API, but in the case of SRM, this API is not available, and tasks can only be restarted by restarting the SRM Driver cluster.

Driver inter-node coordination

Learn about how different Streams Replication Manager Driver instances communicate with each other.

As per the Connect framework, members of a Connect group get updates from the group leader through Kafka, using the Connect group protocol. In some cases, followers also need to be able to communicate with the group leader. For this purpose, the Connect framework introduced a REST API. This API allows followers to push notifications to the leader. This step is necessary when dynamic configuration changes occur in a Connector. In case of SRM, configuration changes happen when new topics and groups are added to the replication. This can happen in two ways:

- New topics and groups appear in the source cluster that conform to the allow and deny lists of the replication.
- Allow and deny lists are added using the srm-control tool.

To support this necessary channel of communication inside SRM, each Driver spins up a REST server per replication flow. With this implementation, each nested Connect worker is a fully functional, as per the original design of Connect. These replication specific REST servers are configurable, for more information, see *Configuring replication specific Kafka Connect REST servers*.

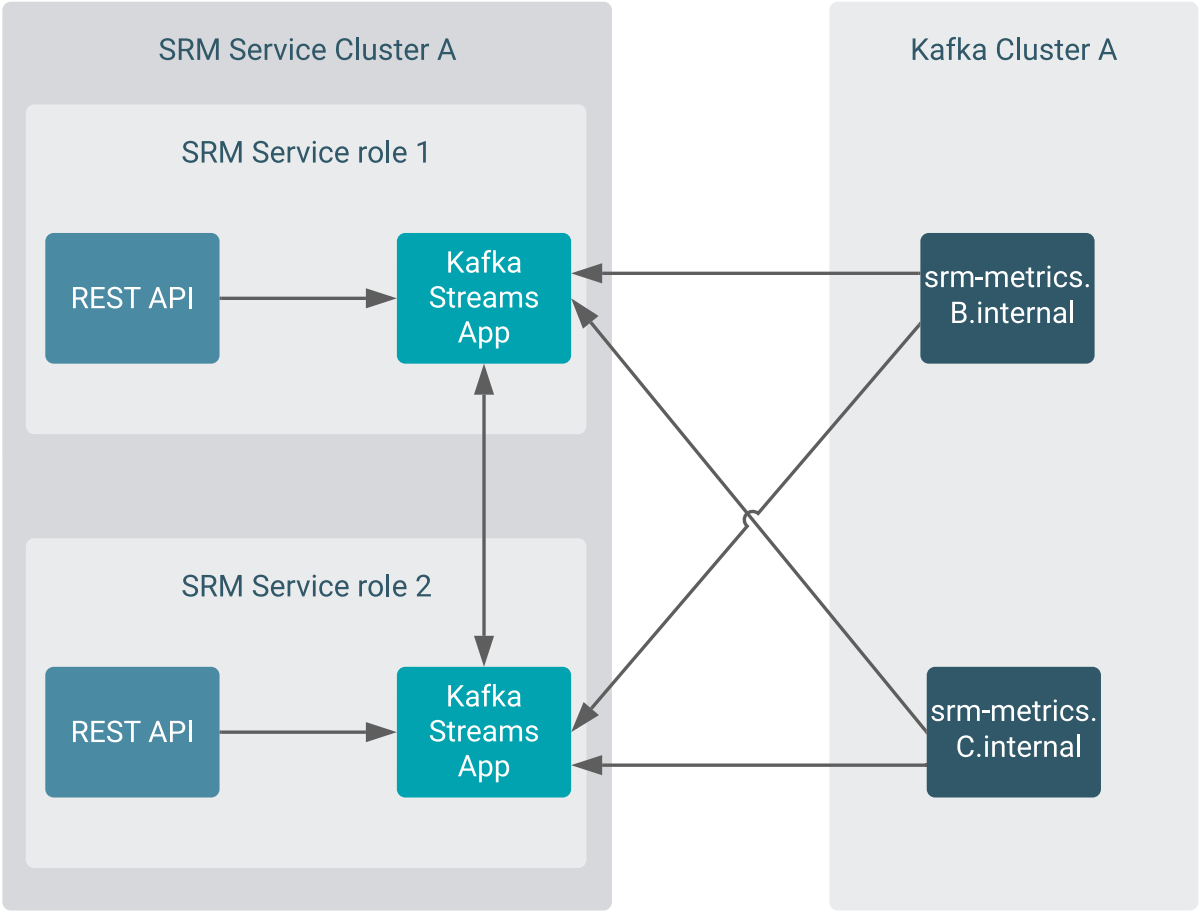
Related Information

Configuring replication specific Kafka Connect REST servers

Streams Replication Manager Service

The Streams Replication Manager Service is responsible for processing the metrics produced by the Streams Replication Manager Drivers. Additionally, it provides a queryable REST API that you can use to monitor and track replications.

Streams Replication Manager Driver roles (SRM Driver) produce raw metrics into the target Kafka clusters. Each replication has a separate raw metric topic. Streams Replication Manager Service roles (SRM Service) run a Kafka Streams application internally, which aggregates the raw metrics. The Streams application ensures that the metrics processing work is load balanced between the SRM Service instances of the same cluster, and that the members of the cluster can coordinate when serving REST API queries.



As a result of this architecture, the SRM Service can only report on replication flows targeting the Kafka cluster which is targeted by the SRM Service.

Using the SRM Service REST API

The SRM Service offers a Swagger UI for exploring and querying the REST API. For a user-friendly UI solution, Streams Messaging Manager (SMM) can be integrated with SRM. When SMM is configured to connect to SRM, the Replications page becomes available on the SMM UI. This page displays the information available on the SRM Service REST API. For more information on how you can integrate the two services see *Configuring SMM for monitoring Kafka cluster replications*.

Related Information

Configuring SMM for monitoring Kafka cluster replications

Understanding Replication Flows

Get familiar with the concept of replications and replication flows and learn more about how they can be set up.

Replication Flows Overview

Get familiar with the concept of replications and replication flows.

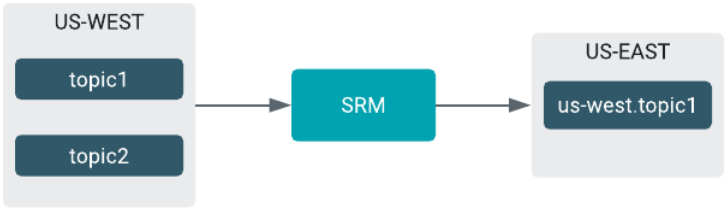
Replication involves sending records from a source cluster to a target cluster. In SRM a replication refers to a source and target cluster pair, the direction in which data is flowing and the topics that are being replicated. Source target cluster pairs can be specified in Cloudera Manager; they are notated source->target. Initially, when source->target pairs are set up they are considered inactive, as no data is being replicated between them. To start replication users need to specify which topics to replicate with the srm-control command line tool.

It is important to understand that replication in SRM is configured independently for each source->target cluster pair. Moreover, configuration is done on a per topic basis. This means that each topic in a source cluster can have a different direction or target that it is being replicated to. A set of topics in the source cluster can be replicated to multiple target clusters while others are being replicated to only one target cluster. This allows users to set up powerful, topic specific replication flows.

The term replication flow is used to specify all replications set up in a system. This document uses the term when referring to the visual representation of SRM replications.

A basic example of a replication flow is when topics are being sent from one cluster to another cluster in a different geographical location. Note that in this example there is only one replication or source->target pair. Moreover, only one of the two topics on the source cluster are being replicated to the target cluster.

Figure 9: Simple Replication Flow Example

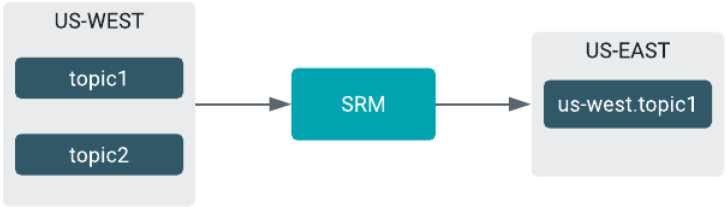


Remote Topics

Learn about SRM's remote topics.

In any replication flow, the selected source topics are replicated to remote topics on the target cluster. Remote topics reference the source cluster via a naming convention. For example, the topic1 topic from the us-west source cluster creates the us-west.topic1 remote topic on the target cluster.

Figure 10: Simple Replication Flow Example



Remote topics can themselves be replicated. In this case, the remote topic references all source and target clusters. The prefix in the name will start with the cluster closest to the final target cluster. For example, the topic1 topic replicated from the us-west source cluster to the us-east cluster and then to the eu-west cluster will be named us-east.us-west.topic1.

Figure 11: Complex Replication Flow Example

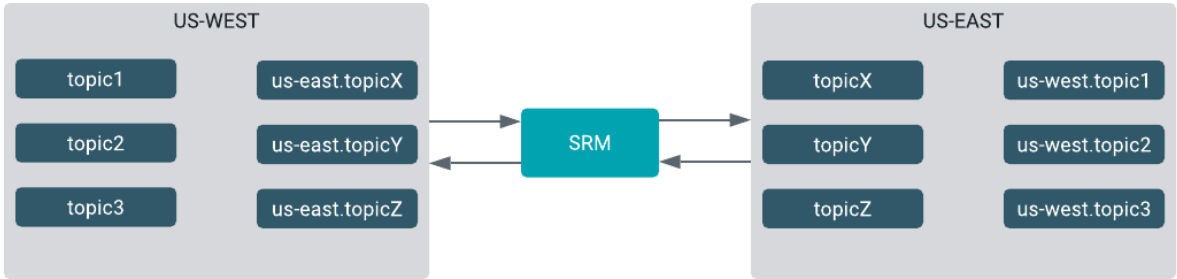
Tip: You might want to have your Kafka consumers read messages from both source and remote topics simultaneously. To achieve this, Kafka consumers should include a wildcard topic name pattern. For example, suppose that you want your consumer to read from topic1 located in us-west and its remote counterpart, us-west.topic1, located in us-east. In such a case, you can use the .*topic1 pattern, which matches any topic that ends with topic1.

Bidirectional Replication Flows

Learn more about bidirectional replication flows.

SRM understands cycles and will never replicate records in an infinite loop. This enables bidirectional replication flows in which clusters are mutually replicated. In this case, records sent to one cluster will be replicated to the other and the other way around. You can configure any number of clusters in this way.

Figure 12: Bidirectional Replication Flow

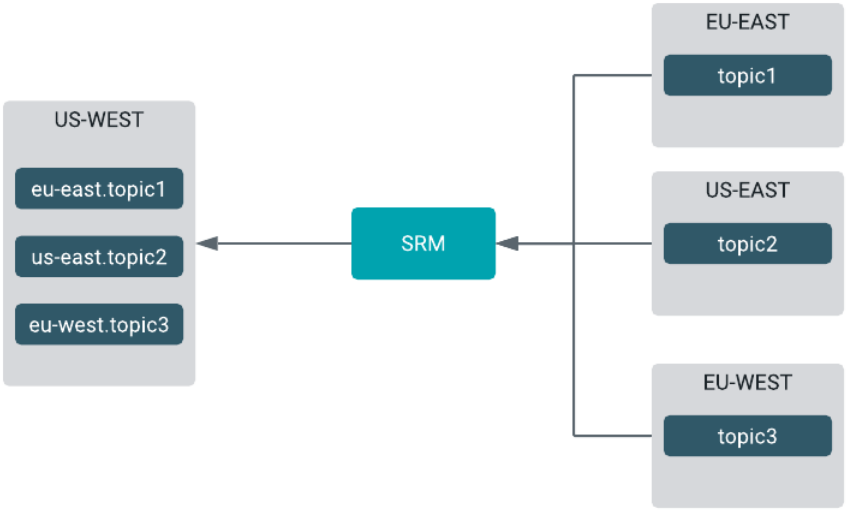


Fan-in and Fan-out Replication Flows

Learn about fan-in and fan-out replication flows.

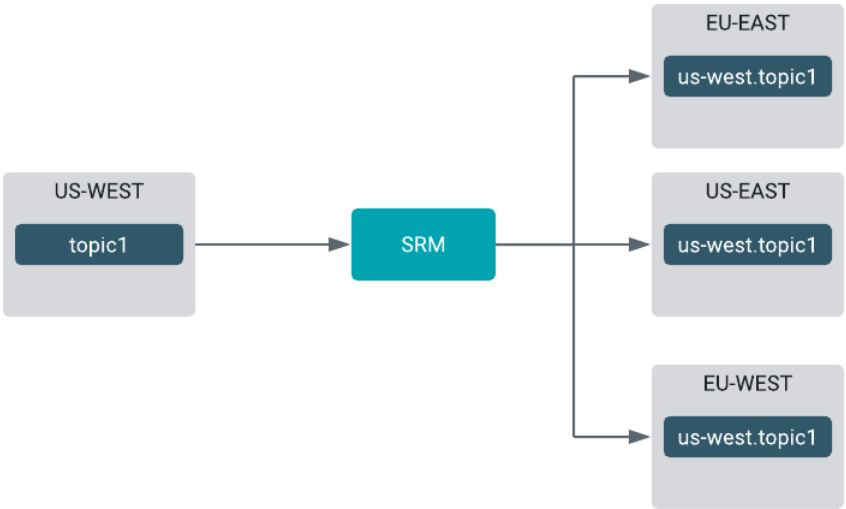
You can construct fan-in replication flows, where records from multiple source clusters are aggregated in a single target cluster.

Figure 13: Fan-in Replication Flow



Similarly, you can construct fan-out replication flows as well, where a single cluster is replicated to multiple target clusters.

Figure 14: Fan-out Replication Flow



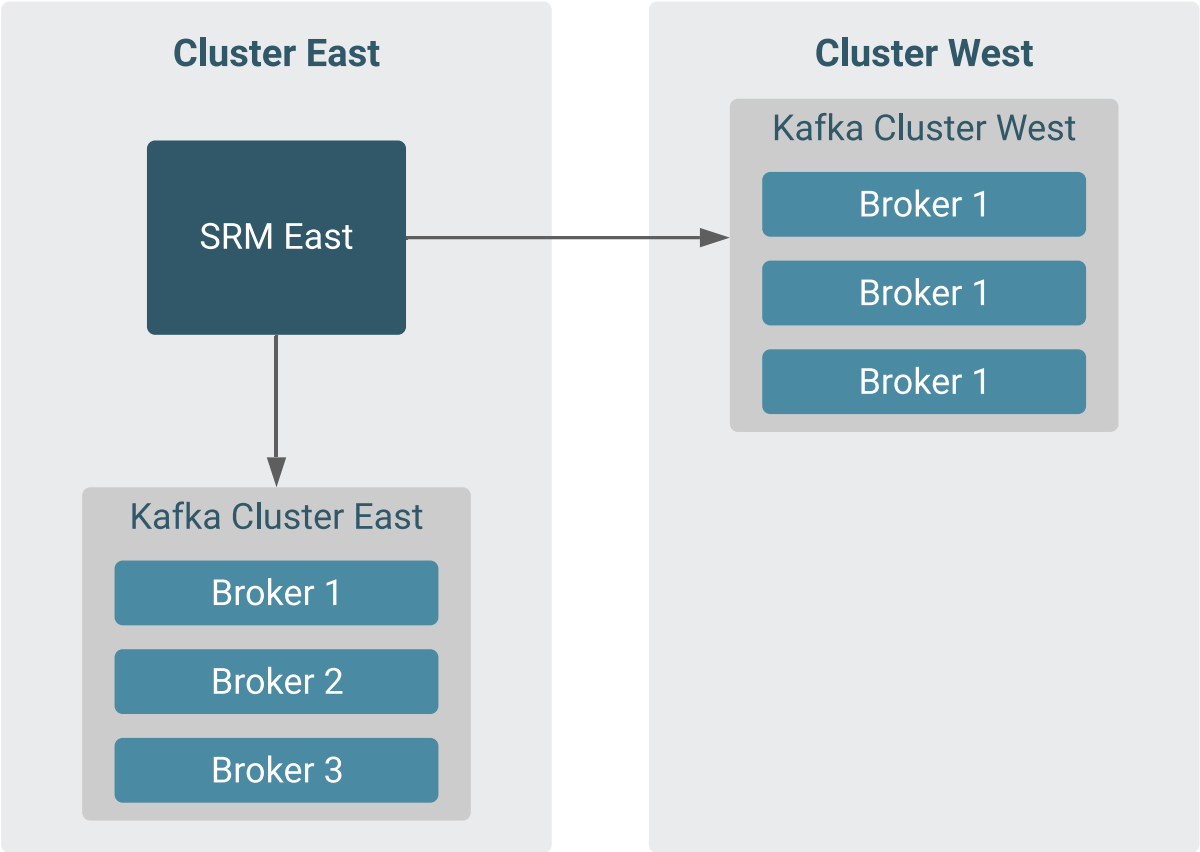
Understanding co-located and external clusters

The Kafka clusters that Streams Replication Manager (SRM) connects to can be categorized into two groups. They can be co-located with or external to SRM. The category dictates how you configure the SRM service and the srm-control tool.

SRM connects to and replicates data between Kafka clusters, which consist of one or more Kafka brokers, deployed on clusters. These Kafka clusters that SRM connects to can be categorized into two groups. They can either be co-located with or external to SRM. Which category a specific cluster falls into is decided based on the relation between that cluster and the SRM service.

A co-located Kafka cluster is the Kafka cluster that is running in the same cluster as the SRM service. Any other Kafka cluster that is remote to SRM either logically or geographically is considered external.

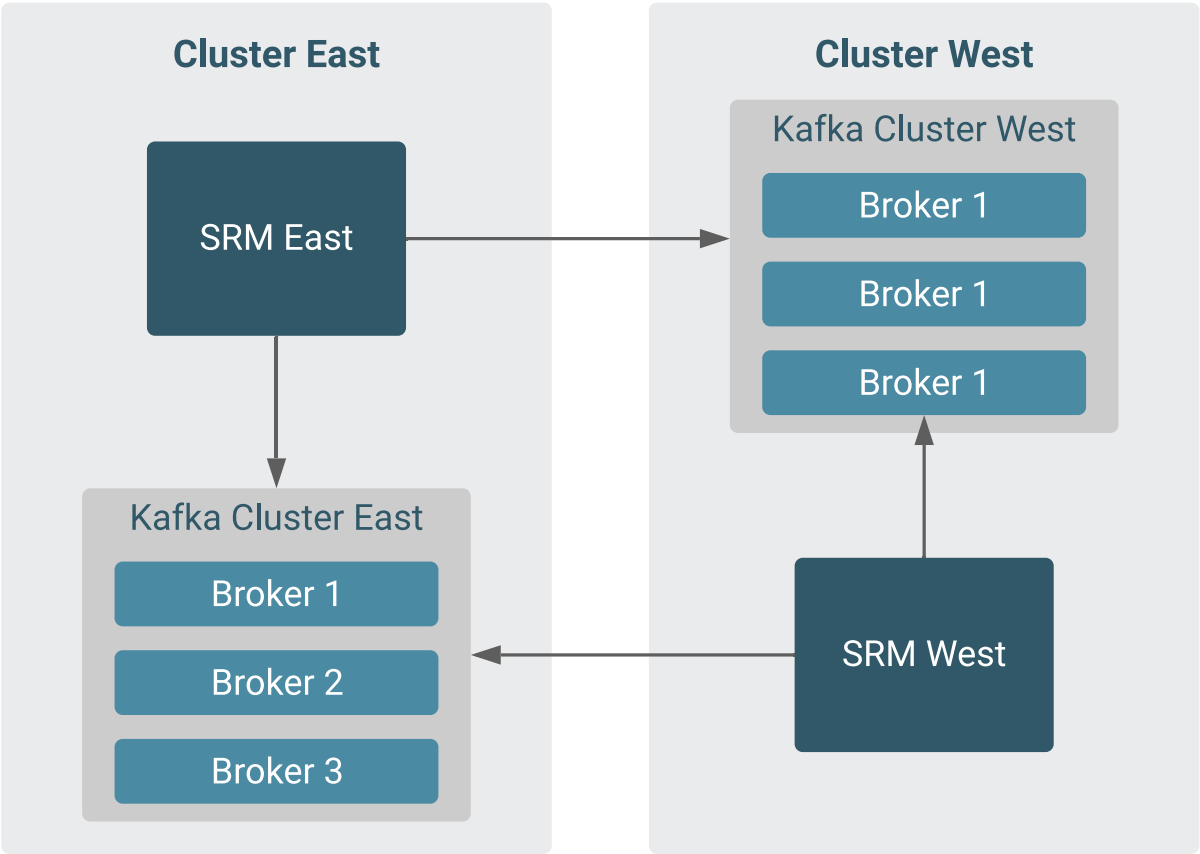
For example, consider the following deployment:



This deployment has two clusters, and both clusters have a Kafka cluster. However, only Cluster East has SRM deployed on it. From the perspective of SRM East, Kafka Cluster East is co-located, while Kafka Cluster West is external.

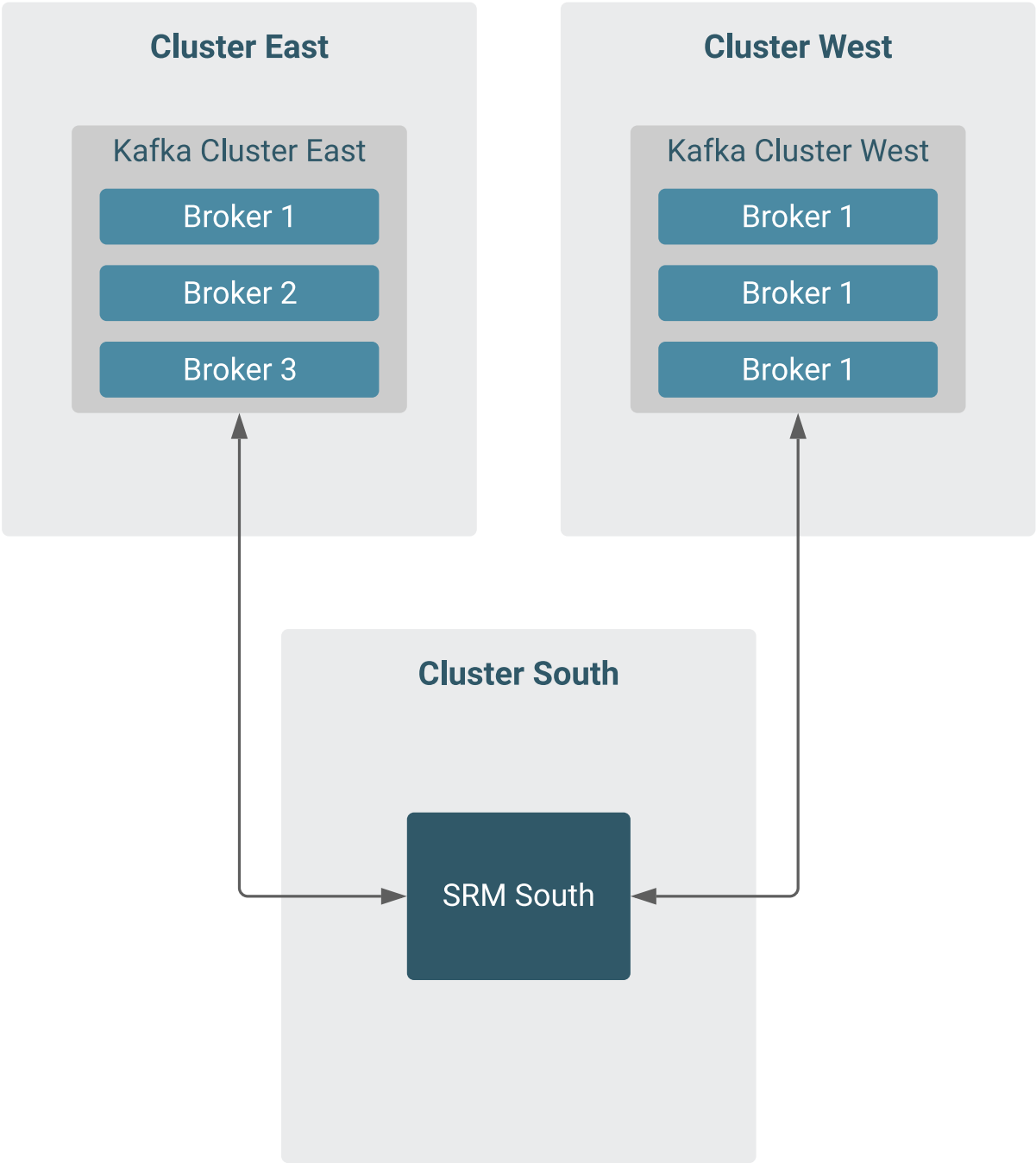
In a more advanced deployment with multiple SRM services, a single Kafka cluster will fall into both categories. From the perspective of a specific SRM service a cluster will be co-located, while for others it will be external.

For example, consider the following deployment:



In this example, both clusters have a Kafka cluster as well as SRM. From the perspective of SRM East, Kafka Cluster East is co-located, Kafka Cluster West is external. From the perspective of SRM West, Kafka Cluster West is co-located, Kafka Cluster East is external.

It is also possible to not have co-located Kafka clusters. For example, consider the following deployment:



In this example, the clusters that have Kafka deployed on them do not have SRM. Instead, data is replicated by an SRM instance deployed on a separate cluster. From the perspective of SRM South, both Kafka Cluster East and West are external, there is no co-located cluster. In a scenario like this, configuration tasks related to the co-located cluster do not need to be completed.

Being able to correctly identify what category a Kafka cluster falls into is important as the category dictates how you configure each SRM service and the srm-control tool. In general, a co-located Kafka cluster requires less configuration than external Kafka clusters. This is because Cloudera Manager is able to automatically pass certain configuration properties about the co-located Kafka cluster to SRM. External Kafka clusters on the other hand must be fully configured and specified manually.

For more information on how to configure and set up SRM see Add Streams Replication Manager to an existing cluster. In addition you can also review any of the configuration examples available in Configuration Examples or Securing Streams Replication Manager.

Understanding SRM properties, their configuration and hierarchy

There are a number of configuration properties that Streams Replication Manager (SRM) accepts, but are not exposed directly in Cloudera Manager. You can configure these properties with the Streams Replication Manager's Replication Configs property. Additionally, these properties can be configured on different levels allowing for granular control over how and when they are applied.

SRM supports various configuration properties. With the exception of adding and enabling replications, all fundamental properties required to set up and start data replication between clusters are available for configuration directly through Cloudera Manager, or are configured automatically by Cloudera Manager.

In addition to these properties, SRM accepts and supports a number of additional properties including Streams Replication Manager specific properties, Kafka Connect worker properties as well as Kafka properties available in the version of Kafka that you are using. These however are not directly available for configuration in Cloudera Manager, and do not have dedicated configuration entries on the UI. Properties like these are configured through the Streams Replication Manager's Replication Configs property.

The configuration properties that you add to Streams Replication Manager's Replication Configs can be set on multiple levels with the help of property prefixes. These prefixes allow you to exercise control over when and where (at what level) a configuration is used by SRM. This gives you fine grained control over how data is replicated.

The prefixes that you can use and the configuration levels they correspond to are as follows:

- Global level - no prefix

  For example:

  ```
  offset.flush.timeout.ms
  ```

- Cluster level - prefixed with cluster alias

  For example:

  ```
  [***ALIAS***].offset.flush.timeout.ms
  ```

- Replication level - prefixed with a replication name

  For example:

  ```
  [***ALIAS***]->[***ALIAS***].consumer.max.poll.records
  ```

Note: All properties in SRM support prefixes, even the ones that you set directly through the Cloudera Manager UI. However, the prefixes for these properties are set automatically in the background based on your overall configuration. Therefore, from a user perspective, prefixes only become significant when you use Streams Replication Manager's Replication Configs.

Properties follow a hierarchy, which is based on what prefix they have. In general, prefixed properties take precedence over non-prefixed properties, and more specific prefixes take priority over less specific ones. That is, replication level properties take precedence over both cluster and global level configurations. Cluster level properties take precedence over global level configurations. Lastly, global level configurations are overridden by both replication and cluster level properties.

Not all prefixes and configuration levels are valid for all properties. For example, most properties related to Kafka client connections (bootstrap servers, aliases, security configurations and so on) are not relevant on a replication level. While they can be set, they will not have an effect.

The following sections give more information on the available configuration levels/prefixes, give examples of what types of properties can be set at each level, as well as recommendations on when and how to use each.

Global level

Global configuration is achieved by adding the property on its own, without a prefix. This is the most generic configuration, with the lowest priority level. Global level configurations can also be regarded as fallback configurations. This is a result of the property hierarchy. SRM will fall back and use a global level configuration if there are no cluster or replication level configurations set for a specific property.

The following types of properties are accepted at this level by SRM:

SRM service level properties

These are SRM-specific configurations that are applied to SRM as a whole. For example, mm.rest.protocol is a property typically set at this level, as it allows configuring the protocol used by all replication specific Connect REST Servers run internally by the SRM Driver to use the same protocol.

Kafka client connection properties

Client connection properties are used by SRM when it connects to a Kafka cluster. Client connection properties can be set on a global level, however, not all connection related properties are compatible with this level. For example, you can set bootstrap.servers on a global level, but because in the majority of cases you will have a unique bootstrap server set for each individual connection, setting this property on a global level is redundant.

As a result, Cloudera recommends that you only configure security related connection properties at this level and only if you have multiple clusters in your deployment that use the same security configuration. For example, you might have a deployment where SRM connects to many Kafka clusters, and all of them use the same cipher suite. In a case like this you can set ssl.cipher.suites at the global level a single time instead of configuring it on a per cluster basis.

Kafka Connect worker properties

These configurations are applied to all Kafka Connect workers run internally by the SRM Driver. Any Connect worker property is supported at this level as long as it uses one of the following prefixes:

- offset.storage
- config.storage
- status.storage
- key.converter
- value.converter
- header.converter
- task
- worker
- listeners.https

For example, you can configure offset.storage.replication.factor at this level to specify the replication factor of the internal topic used to track the source offsets of SRM in all target clusters.

Connector properties

These configurations are applied to all Connectors in all replications run internally by the SRM Driver. For example, you can configure emit.heartbeats.enabled at this level to control heartbeat emission on all replications.

Cluster level

Cluster level configuration is achieved by prefixing the configuration property with a cluster alias specified in Streams Replication Manager Cluster alias. Cluster level configurations have a higher priority than global configurations. As a result, cluster level configurations take precedence over global configurations.

The following types of configurations are accepted at this level by SRM:

Kafka client connection properties

These configurations are used by SRM to connect to the Kafka cluster identified by the alias. SRM is capable of generating connection configurations securely for most use-cases and Kafka clusters either through automation, or with Kafka Credentials (AdministrationExternal AccountsKafka Credentials). Cloudera recommends that you only use automation or Kafka credentials to configure these properties.

For advanced use-cases, or when a specific connection configuration property is not available through Kafka credentials, the cluster prefix can be used to specify the configuration property. For example, the sasl.login.callback.handler.class property cannot be set using Kafka credentials. If in your deployment you are using a third-party callback handler and need to specify it for SRM, you can do so by adding sasl.login.callback.handler.class with the appropriate cluster prefixes to Streams Replication Manager Replication Configs.

Kafka Connect worker properties

These configurations are applied to Kafka Connect workers which correspond to a replication targeting the Kafka cluster identified by the alias prefix. On this level, all Connect worker configurations are supported. For example, you can use offset.flush.timeout.ms to increase the flush timeout to support high-traffic replication flows.

Replication level

Replication level configuration can be achieved by prefixing the configuration property with the name of the replication. Replication level configurations have a higher priority than cluster or global level configurations. As a result, replication level configurations take precedence over both cluster and global configurations.

The following types of configurations are accepted at this level by SRM:

Connect worker properties using the cluster->cluster.worker.prefix

These properties are applied to the Kafka Connect workers which correspond to the replication identified by the prefix. On this level, all Connect worker configurations are supported. For example, offset.flush.timeout.ms can be specified to increase the flush timeout to support high-traffic replication flows.

Connector properties

These configurations are applied to all Connectors in the replication identified by the prefix.

In addition to the replication specific Connector configurations, you can also tweak the consumers and producers that are used within a specific replication. For example, by using the cluster->cluster.consumer. prefix, any consumer property can be specified to tweak the consumers used in the replication. Similarly, the cluster->cluster.producer.override. prefix can be used to override producer configurations in the replication flow.

Related Information

Configuring properties not exposed in Cloudera Manager