Cloudera Runtime 7.3.2

Streams Streams Replication Manager Overview

Date published: 2019-08-22
Date modified: 2026-03-31



https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2026. 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 7

Streams Replication Manager use case architectures 7
- Highly available Kafka architectures 7
- Cluster migration architectures 10

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
  - Remote Querying 21

Streams Replication Manager monitoring and metrics 24

Streams Replication Manager Service REST API 27

Streams Replication Manager replication flows and replication policies 28

Streams Replication Manager remote topic discovery 32

Automatic group offset synchronization 33

Understanding co-located and external clusters 34

Understanding Streams Replication Manager properties, their configuration and hierarchy 38

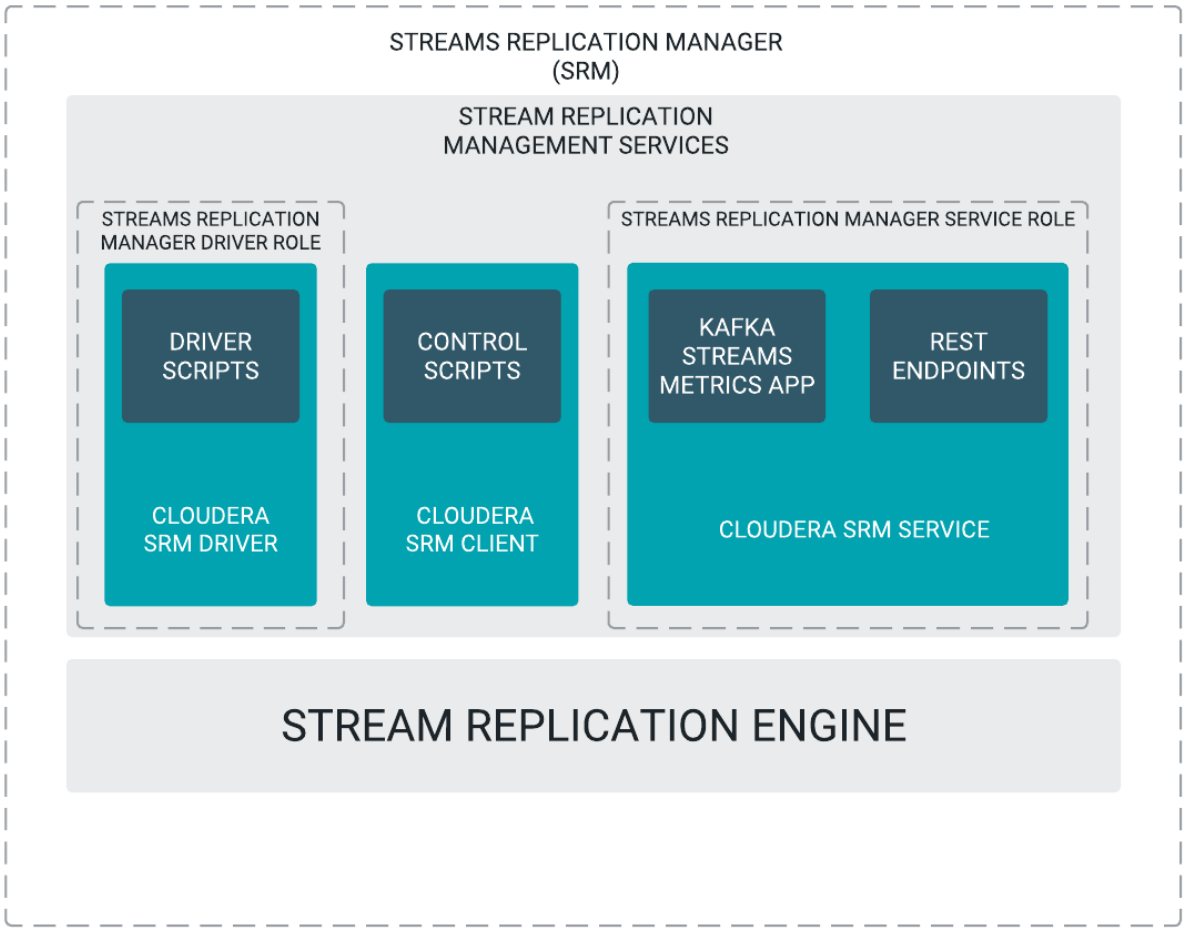
Reverse Checkpointing 42

Overview

Get familiar with Streams Replication Manager and its components.

Streams Replication Manager is an enterprise-grade replication solution that enables fault tolerant, scalable and robust cross-cluster Kafka topic replication. Streams Replication Manager provides the ability to dynamically change configurations and keeps the topic properties in sync across clusters at high performance. Streams Replication Manager also delivers custom extensions that facilitate installation, management and monitoring making Streams Replication Manager 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 Streams Replication Manager Driver

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

Cloudera Streams Replication Manager Client

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

Cloudera Streams Replication Manager Service

The Cloudera Streams Replication Manager 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

Streams Replication Manager has the following main features.

Remote topics

Streams Replication Manager replicates Kafka topics from source to target clusters. Remote topics are the replica topics located in target clusters. Remote topics and their source topics are tracked internally by Streams Replication Manager. Additionally, remote topics are by default prefixed with the name (alias) of the source cluster. The naming convention for remote topics is configurable and is determined by the replication policy that is currently in use. For more information, see *Replication flows and replication policies*.

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, Streams Replication Manager 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. Streams Replication Manager 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

Streams Replication Manager 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

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

Centralized configuration for multi-cluster environments

Streams Replication Manager 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 Streams Replication Manager Service collects and aggregates Kafka replication metrics and make them available through a REST API. This REST API is used by Streams Messaging Manager 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.

Replication policies

The replication policy used by Streams Replication Manager defines the basic rules of how Streams Replication Manager replicates data. Streams Replication Manager ships with two replication policies that are designed for different use cases. These are the `DefaultReplicationPolicy`, which uses topic name prefixes to provide replication loop detection, and the `IdentityReplicationPolicy`, which mimics the behavior of MirrorMaker 1 and provides prefixless replication. Both policies also support the monitoring features provided by the Streams Replication Manager Service.

In addition to the Cloudera provided policies, custom developed replication policies can be used. Developing and using your own replication policy enables you to gain full control over how Streams Replication Manager replicates data.



Important: Cloudera provides limited support for deployments that use custom replication policies. Additionally, some key features including replication monitoring with the Streams Replication Manager Service will not work if a custom replication policy is in use. Whenever possible, Cloudera recommends that you use the `DefaultReplicationPolicy`.

For more information, see *Replication flows and replication policies*.

Remote topic discovery and tracking

Streams Replication Manager uses an internal Kafka topic to track which topics are being replicated in a replication flow. This enables the Streams Replication Manager Service to filter topics from monitoring if they are no longer being replicated. Additionally, this feature enables replication monitoring with the Streams Replication Manager Service even if prefixless replication is being used.

Related Information

Monitoring Kafka cluster replications using Streams Messaging Manager

Streams Replication Manager replication flows and replication policies

Replication flows and replication policies

Main Use Cases

Learn about the main use cases of Streams Replication Manager.

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.

Streams Replication Manager use case architectures

You can deploy Streams Replication Manager in many different ways and for many different use cases. The following collects example use case architectures where Streams Replication Manager is used to achieve high availability or used for cluster migration.

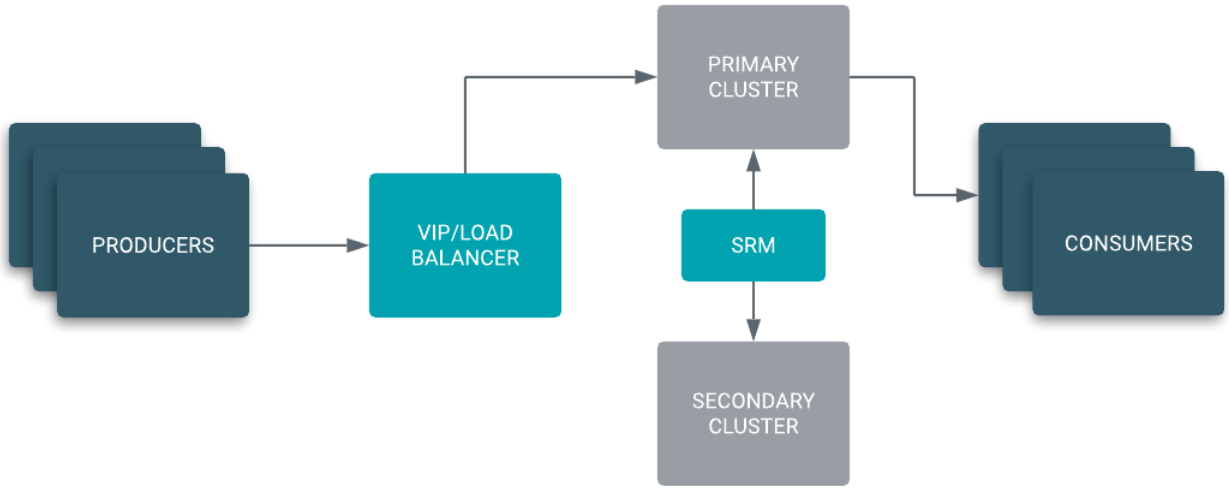
Highly available Kafka architectures

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

Active/Standby architecture

In an Active/Standby scenario, you set up two Kafka clusters and configure Streams Replication Manager 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.

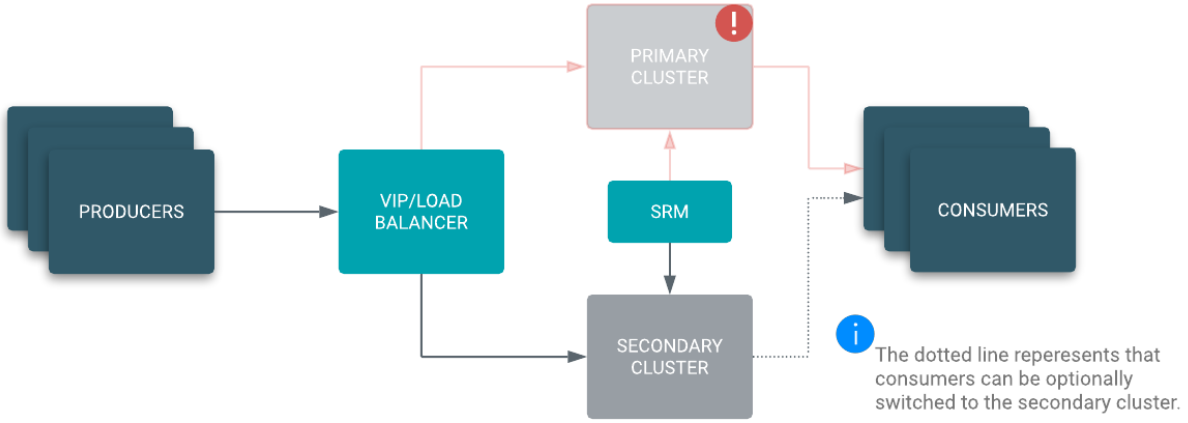
Figure 2: Active/Standby standard operation



In case of a disaster, the VIP or load balancer directs the producers to the secondary cluster. You can easily migrate your consumer groups to start reading from the secondary 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, Streams Replication Manager automatically takes care of synchronizing both clusters making failback seamless.

Figure 3: Active/Standby cluster failure



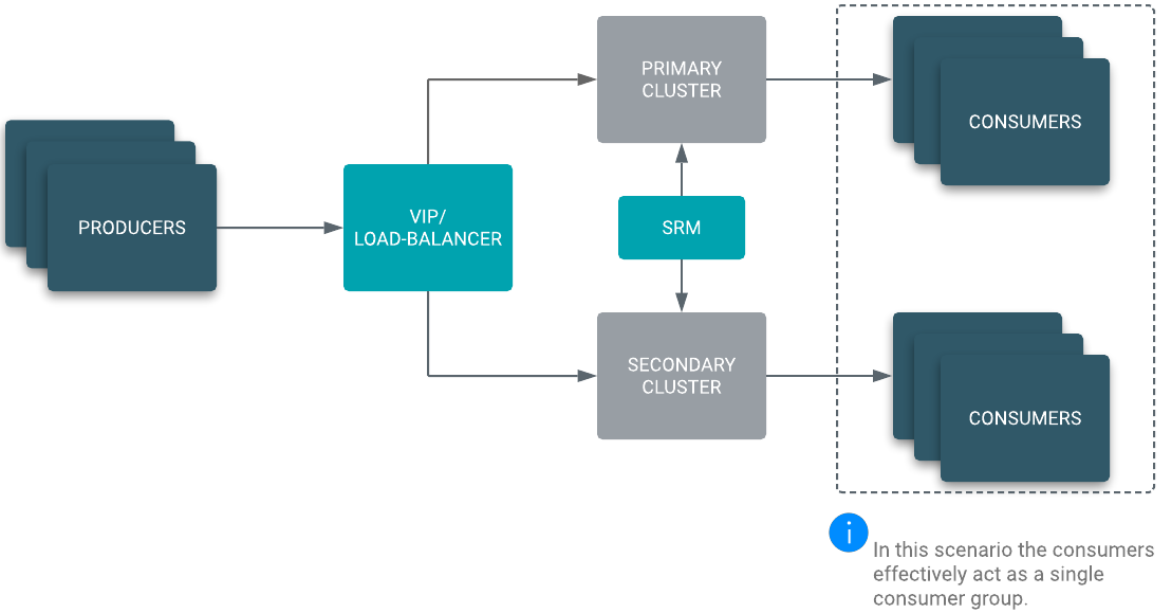
Implementing an Active/Standby architecture is the logical choice when an existing disaster recovery site with established policies is already available, and your goals include not losing ingest capabilities during a disaster and having a backup in your disaster recovery site.

Active/Active architecture

In an Active/Active scenario, your producers can be load balanced to either your primary or secondary cluster. Streams Replication Manager 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 zero downtime and hands-off failover in case of a disaster. Once the primary cluster is back online, Streams Replication Manager automatically takes care of synchronizing both clusters and your primary consumer group resumes processing messages.

Figure 4: Active/Active architecture

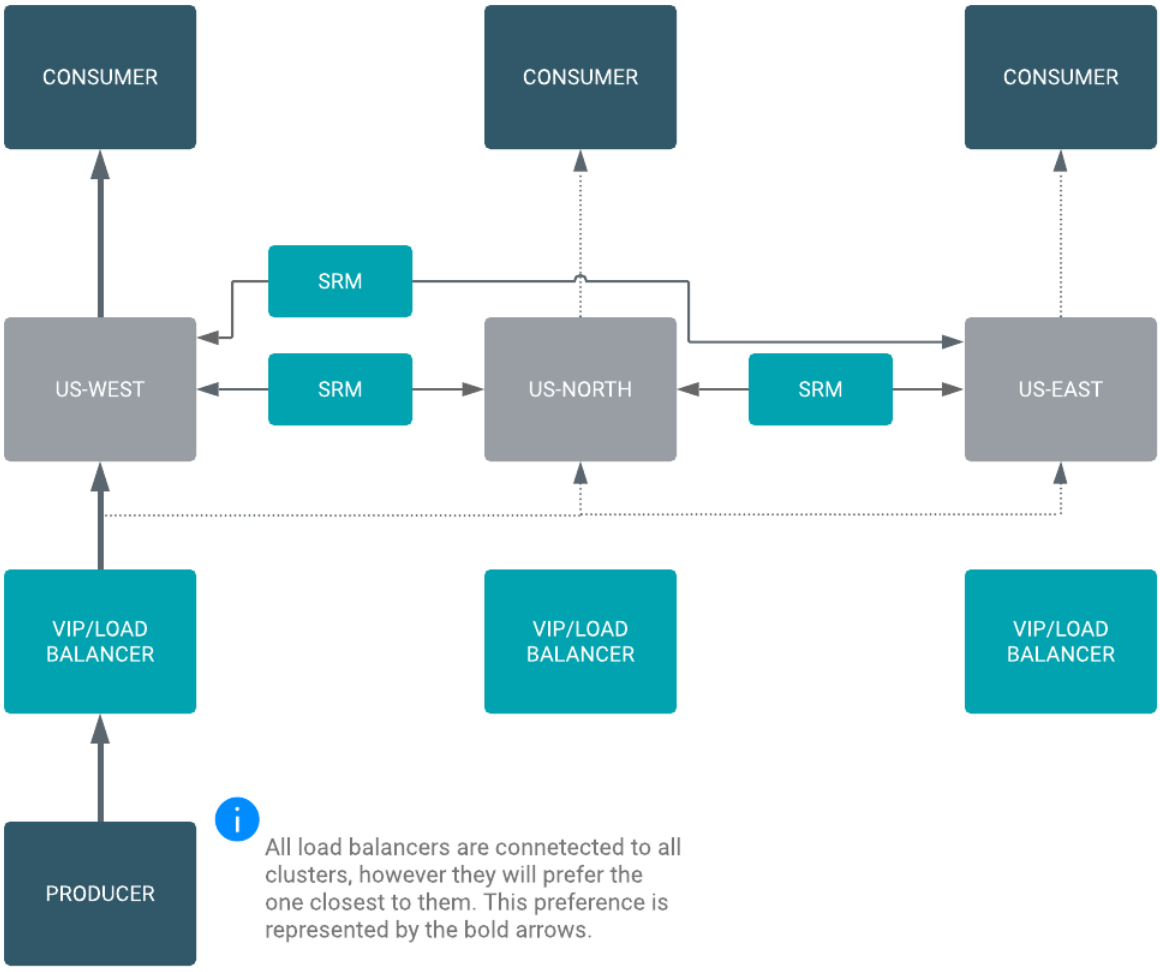


Cross data center replication

Certain applications not only require local high availability within one Data Center (DC) or one Availability Zone (AZ), but have to be highly available across DCs as well. You can use Streams Replication Manager to set up replication between Kafka clusters in different DCs. This results in messages being available to consumers in each of your DCs.

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

Figure 5: Cross DC replication architecture



Cluster migration architectures

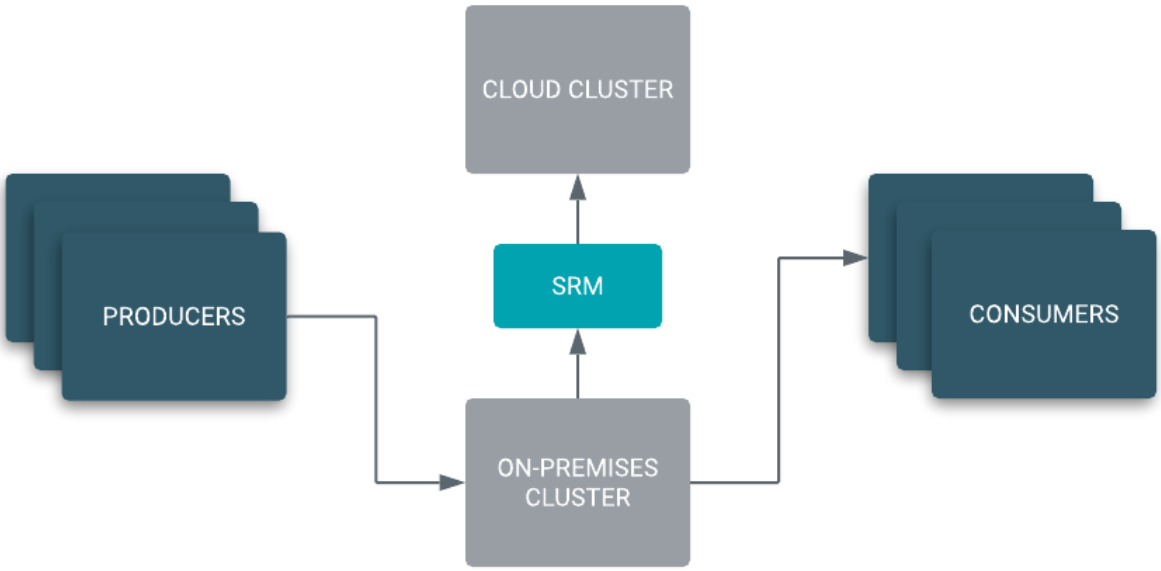
In addition to high availability use cases, Streams Replication Manager can be used for cluster migration and aggregation scenarios as well.

On-premise to cloud and Kafka version upgrade

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 Streams Replication Manager, 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



Aggregation for analytics

Streams Replication Manager can be used to aggregate data from multiple streaming pipelines, possibly across multiple data centers (DC), to run batch analytics jobs that provide a holistic view across the enterprise. For aggregation type architectures Cloudera recommends that you use the `IdentityReplicationPolicy`, which provides prefixless replication. When this replication policy is in use, remote (replicated) topics names are not altered. As a result, you can easily aggregate multiple topics from multiple clusters into one. For more information, see *Replication flows and replication policies* or *Enabling prefixless replication*.

Figure 7: Aggregation for Analytics

Related Information

Replication flows and replication policies

Enabling prefixless replication

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

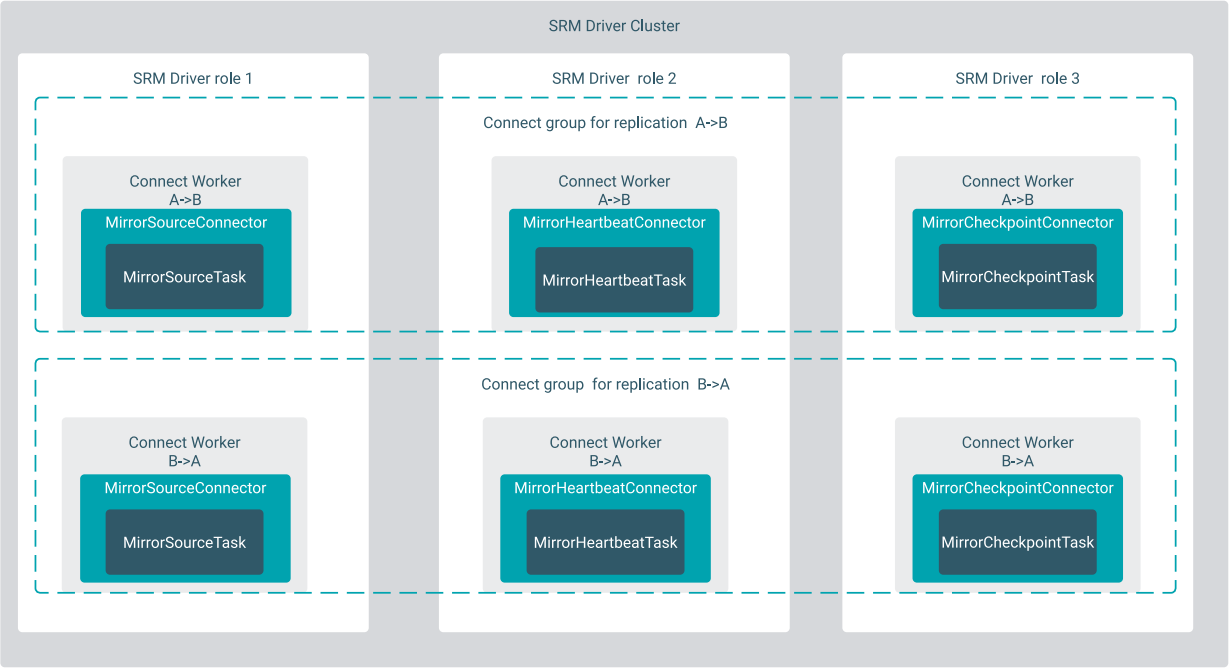
Streams Replication Manager Driver

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 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 Streams Replication Manager Driver must connect to multiple Kafka clusters. The Streams Replication Manager Driver achieves this by wrapping multiple Connect workers in a single driver process. Specifically, for each possible replication between the configured clusters, the Streams Replication Manager 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. Afterward, 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 Streams Replication Manager Driver High Availability is in use, the Connect workers that are tied to a specific replication, but are running in different Streams Replication Manager 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.



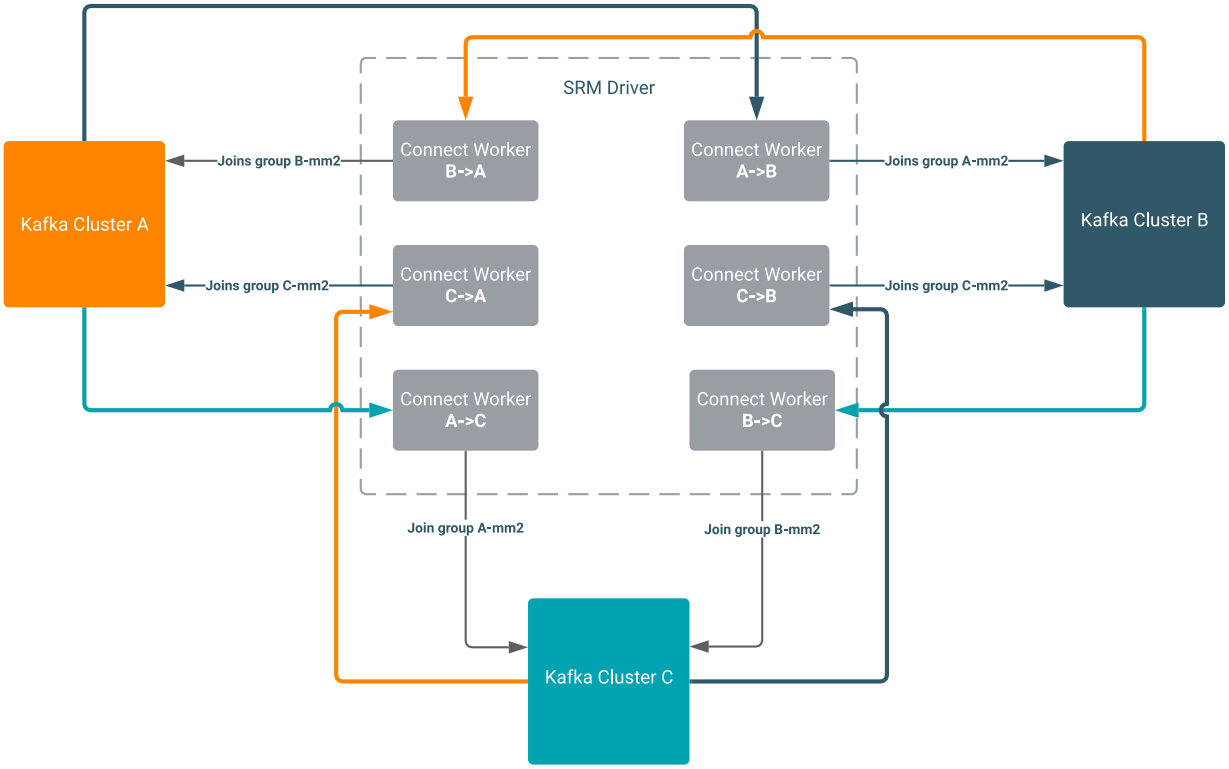
Connect workers

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

The Streams Replication Manager Driver role wraps multiple Connect workers in its process. Each Connect worker corresponds to a possible replication. 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 by default 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 replications, 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.

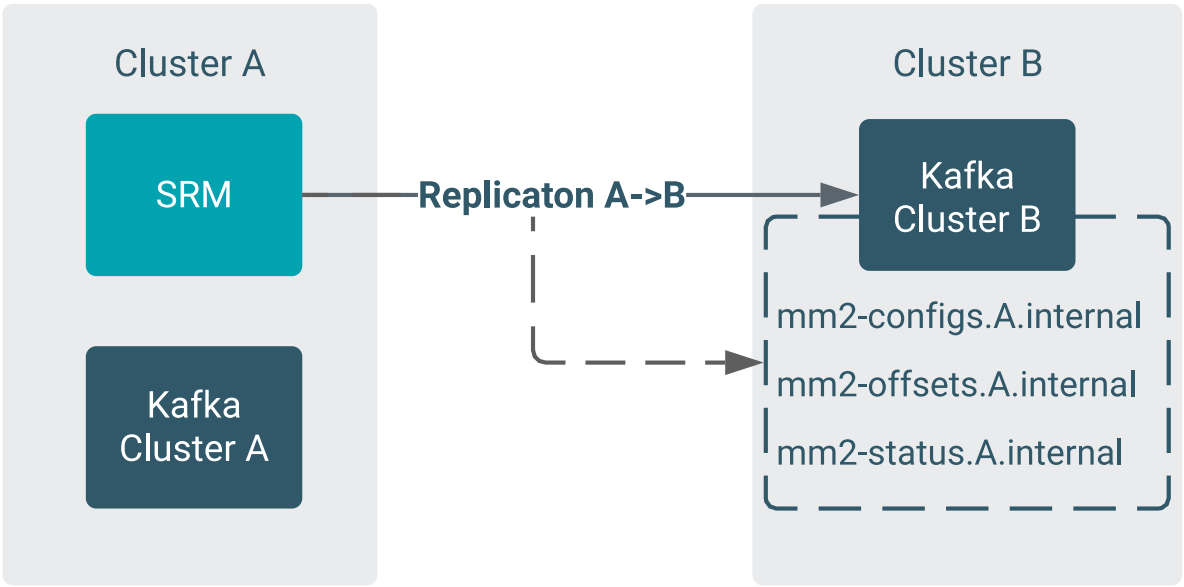
In certain scenarios however, some of the Connect workers set up for disabled replications are unnecessary. In such a case, you can configure Streams Replication Manager to fully deactivate disabled replications that are unutilized. This can be done by disabling heartbeat emission for these replications. If heartbeat emission is disabled, the Connect worker and MirrorHeartbeatConnector are not created. As a result, the replication associated with that worker is fully deactivated. This can help in minimizing the performance overhead caused by unnecessary Connect workers. For more information on how disable heartbeat emission, see *Configuring Streams Replication Manager Driver heartbeat emission*.

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

Streams Replication Manager 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 Streams Replication Manager configuration at startup. Losing the data does not cause issues for Streams Replication Manager after the service is restarted.

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

  Stores the committed source offsets of Streams Replication Manager. Streams Replication Manager 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 Streams Replication Manager 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 Streams Replication Manager after the service is restarted.

All three internal topics are created by Streams Replication Manager 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, Streams Replication Manager 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

Configuring Streams Replication Manager Driver heartbeat emission

Connectors

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

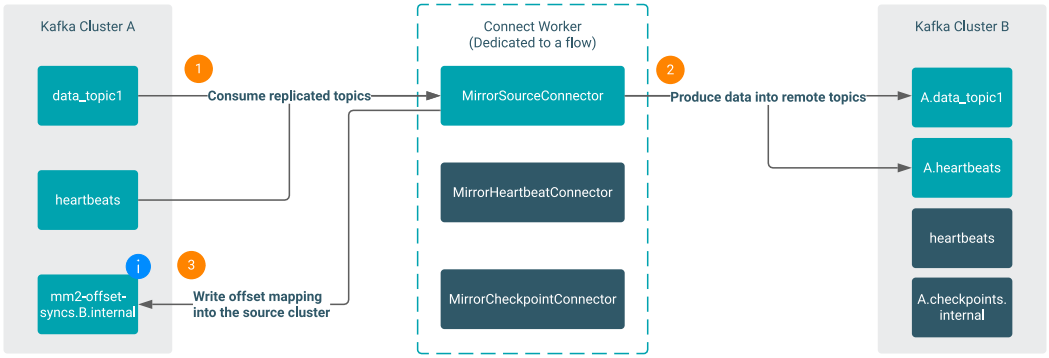
The Streams Replication Manager Driver role employs multiple Connect workers to execute replication, one per each replication. Inside the Connect workers, three separate Connector implementations are utilized. These are the following:

- 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.

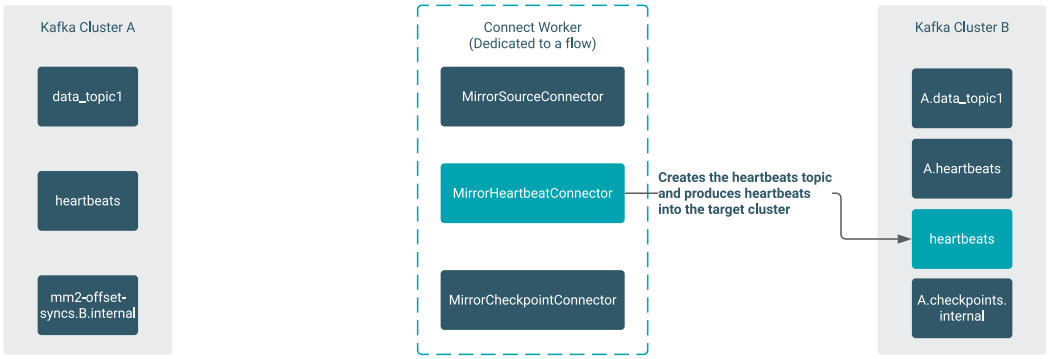
In addition to replicating data, the MirrorSourceConnector also manages the offset sync topic in the source cluster. When production into the target cluster is successful, the mapping between the source offset and the target offset is written into an offset sync topic in the source cluster. The offset 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 Streams Replication Manager Service to discover configured replications.

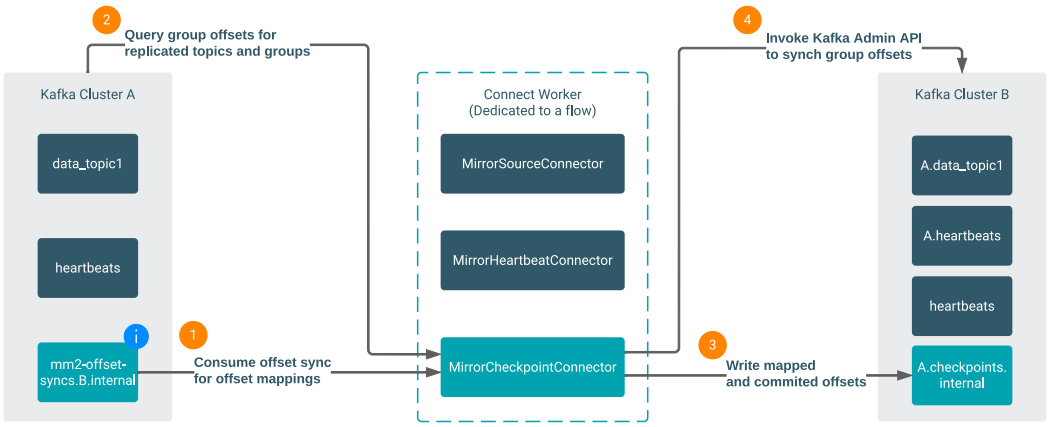


MirrorCheckpointConnector

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 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 Streams Replication Manager. These task instances provide a way to parallelize replication work. After the Connectors create the task configurations, the Connect framework distributes the tasks between the Streams Replication Manager Drivers. More specifically, between the Connect workers running in the Streams Replication Manager Drivers corresponding to a specific replication flow. The task instances are as follows:

MirrorSourceTask

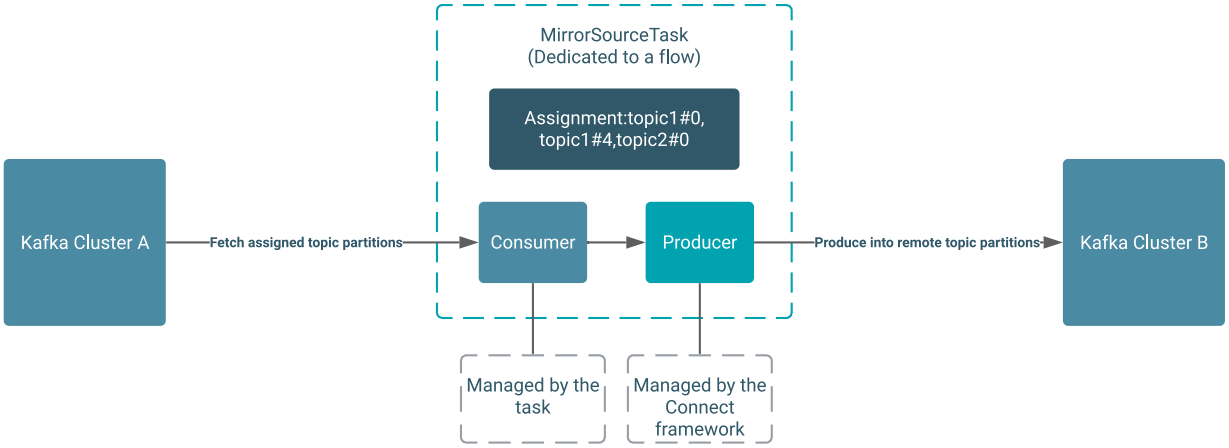
The MirrorSourceTask is created by the MirrorSourceConnector. It is responsible for executing data replication. A MirrorSourceTask wraps a consumer and a producer instance. The consumer instance is managed by the task. The producer instance is managed by the Connect framework.

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 Streams Replication Manager. 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 Streams Replication Manager. 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 Streams Replication Manager 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 Streams Replication Manager, this API is not available, and tasks can onle restarted by restarting the Streams Replication Manager 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 Streams Replication Manager, 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 Streams Replication Manager, each Driver spins up a REST server per replication flow. With this implementation, each nested Connect worker is 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*.

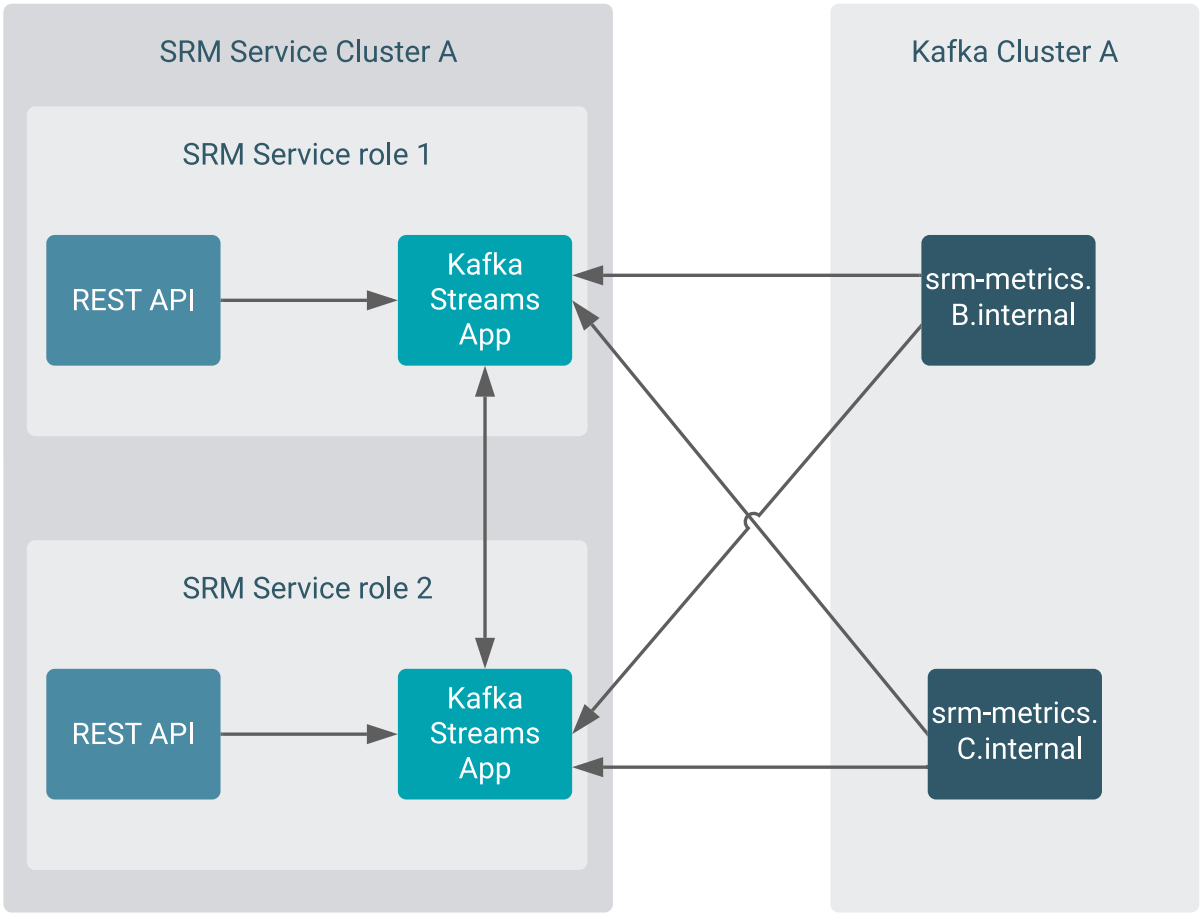
Related Information

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 produce raw metrics into the target Kafka clusters. Each replication has a separate raw metric topic. Streams Replication Manager Service roles 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 Streams Replication Manager 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 Streams Replication Manager Service can only report on replication flows targeting the Kafka cluster which is targeted by the Streams Replication Manager Service.

Multi cluster targeting

The Streams Replication Manager Service is capable of targeting multiple Kafka clusters. In this case, a Streams Replication Manager Service instance spins up a Kafka Streams application for each target Kafka cluster. This way, the Streams Replication Manager Service is able to report on all replication flows targeting any of the configured target clusters.

Remote Querying

A Streams Replication Manager Service cluster can be configured to target and gather metrics from multiple clusters (multi cluster targeting). However, a setup like this can result in heavily loaded Service roles, which might not be suitable for your deployment. To circumvent this issue, the Streams Replication Manager Service is capable of connecting to and communicating with other, remote Streams Replication Manager Services to fetch their metrics. This is called Remote Querying. Using Remote Querying makes it possible to designate a Streams Replication Manager Service cluster in your deployment to act as a monitoring gateway. The designated Streams Replication Manager Service cluster then can be used to monitor all clusters and replications in your deployment. For more information, see *Remote Querying*.

Using the Streams Replication Manager Service REST API

The Streams Replication Manager Service offers a Swagger UI for exploring and querying the REST API. For a user-friendly UI solution, Streams Messaging Manager can be integrated with Streams Replication Manager.

When Streams Messaging Manager is configured to connect to Streams Replication Manager, the **Replications** page becomes available on the Streams Messaging Manager UI. This page displays the information available on the Streams Replication Manager Service REST API. For more information on how you can integrate the two services, see *Integrating Streams Replication Manager with Streams Messaging Manager*.

Related Information

Integrating Streams Replication Manager with Streams Messaging Manager

Remote Querying

Remote Querying in Streams Replication Manager refers to the Streams Replication Manager Service's capability of querying other, remote Streams Replication Manager Services to fetch the remote cluster replication metrics. This allows users to monitor all replications of a deployment that has multiple instances of Streams Replication Manager through a single Streams Replication Manager Service.

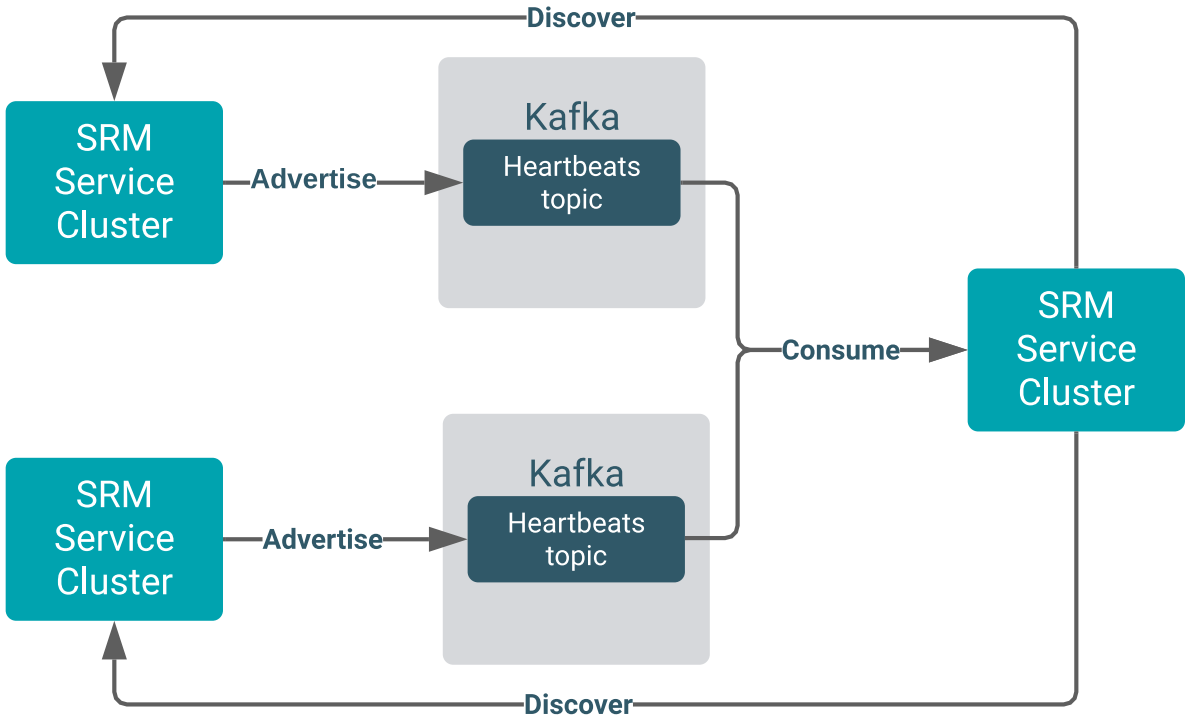
Overview

The Streams Replication Manager Service role gathers, aggregates, and exposes metrics related to cluster replications. While a single cluster of Streams Replication Manager Service roles (Streams Replication Manager Service cluster) can be configured to target and gather metrics from multiple clusters, a setup like this can result in heavily loaded Service roles, which might not be suitable for your deployment. Instead, you can choose to have a single Streams Replication Manager Service cluster connect to other, remote Streams Replication Manager Service clusters and fetch metrics from them. This is called Remote Querying.

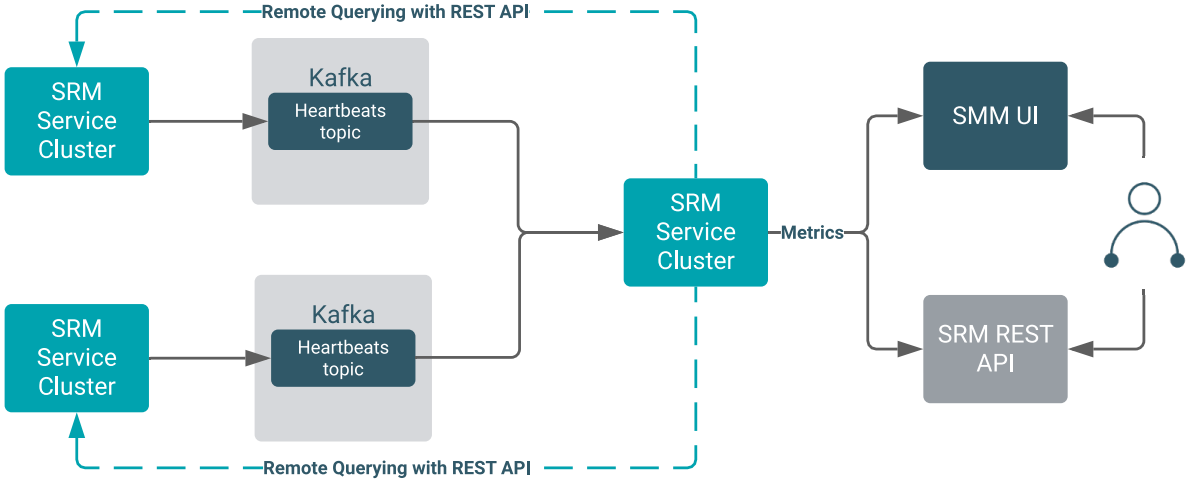
Using Remote Querying makes it possible to designate a Streams Replication Manager Service cluster in your deployment to act as a monitoring gateway. The designated Streams Replication Manager Service cluster can then be used to monitor all clusters and replications in your deployment. This way, a single Streams Replication Manager Service cluster can provide you with information on all clusters and replications. In addition, if you have Streams Messaging Manager integrated with the Streams Replication Manager Service cluster acting as the gateway, information regarding all replications will be available in that Streams Messaging Manager instance's UI.

How it works

Remote Streams Replication Manager Service clusters are discovered through Kafka. Streams Replication Manager Service clusters advertise themselves through their target Kafka cluster by writing data into a heartbeats topic. The information advertised is the Service role's protocol, host, port and root API path. When Remote Querying is configured for a specific Streams Replication Manager Service cluster, that Streams Replication Manager Service cluster connects to the specified external Kafka clusters, consumes the heartbeats topics, and based on the advertised information, discovers the remote Streams Replication Manager Service clusters.



Following discovery, a Streams Replication Manager Service cluster can cooperate with its remote counterparts and fetch the metrics related to remote replications. These metrics can then be queried using the Streams Replication Manager REST API, or viewed on the Streams Messaging Manager UI.



Data locality

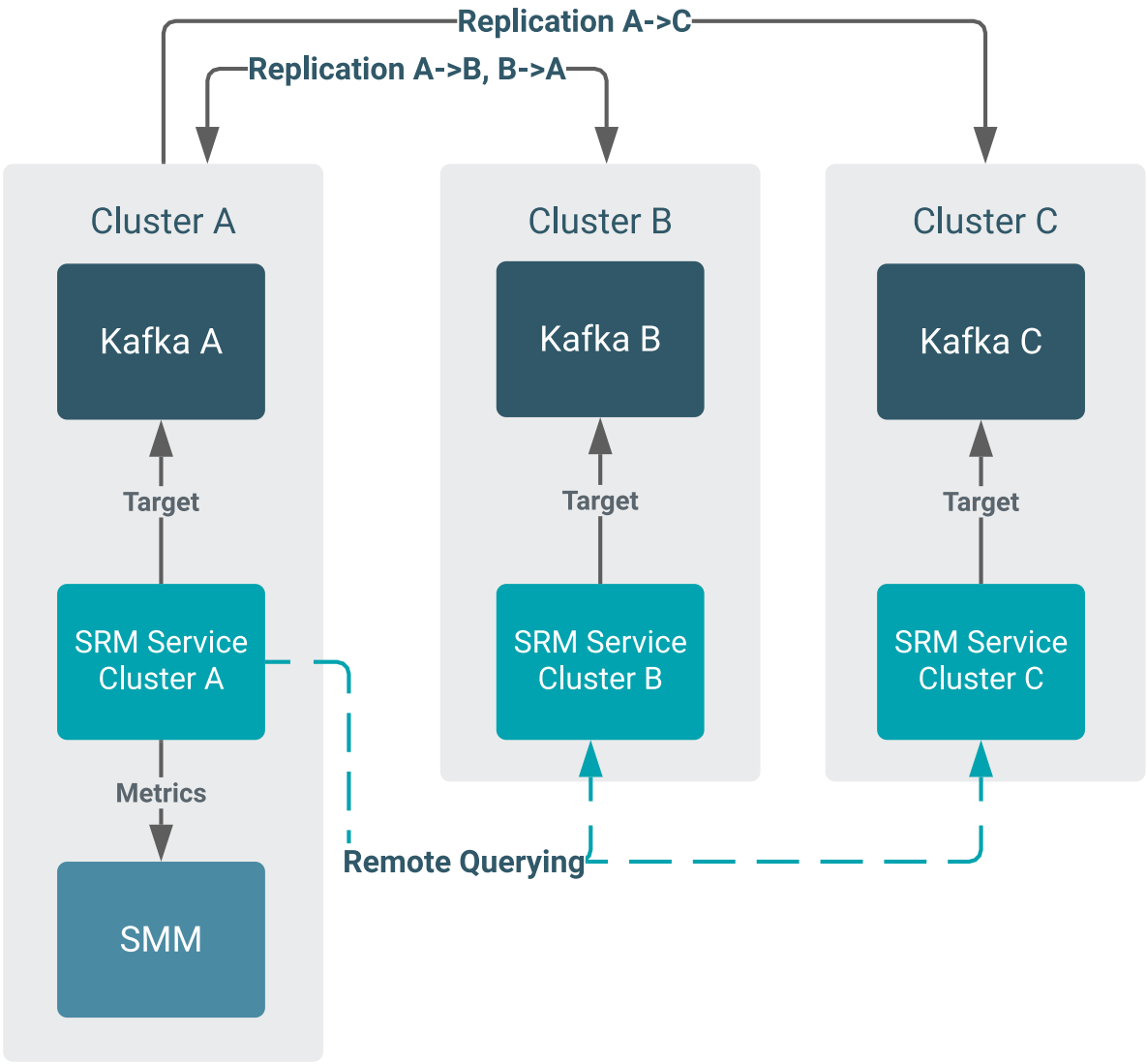
When the feature is enabled, all metrics are still processed locally. Each Streams Replication Manager Service cluster processes the metrics of its target Kafka cluster only. The Streams Replication Manager Service cluster configured to be the gateway does not take over and process the metrics of the remote Streams Replication Manager Service clusters. It only communicates with the remote Streams Replication Manager Service clusters to fetch and then serve their metrics.

However, because metrics processing remains local, when you enable the feature, additional traffic is generated between the gateway and remote Streams Replication Manager Service clusters. It is important that you take this into consideration especially if one or more of your Streams Replication Manager installations are located in a public

cloud environment. For more information on the amount of data generated, see *Streams Replication Manager Service data traffic reference*.

Remote Querying example

Consider the following deployment:



There are three clusters, cluster A, B, and C. All clusters have Kafka and Streams Replication Manager deployed on them. Cluster A has Streams Messaging Manager installed as well. Bidirectional replication is happening between Cluster A and Cluster B. Additionally, unidirectional replication is set up from Cluster A to Cluster C. Each Streams Replication Manager Service cluster is targeting its co-located Kafka.

In this scenario, Remote Querying is configured and enabled for Streams Replication Manager Service cluster A. This enables you to monitor all replications in the deployment using Streams Replication Manager Service cluster A. Additionally, information regarding all replications can be viewed in Streams Messaging Manager deployed in Cluster A.

Without Remote Querying, Streams Messaging Manager would only be able to display replications that are targeting Cluster A. If you wanted to monitor any other replications, you would need to manually query each Streams Replication Manager Service cluster separately using the REST API, or set up separate instances of Streams Messaging Manager on each of the clusters.

Related Information

Streams Replication Manager Service data traffic reference

Streams Replication Manager monitoring and metrics

Streams Replication Manager metrics can give you insight into the status of Streams Replication Manager and the replication process as a whole. Learn how Streams Replication Manager metrics are collected, processed, and exposed on the Streams Replication Manager Service REST API.

Important: While Streams Replication Manager monitoring using the Streams Replication Manager Service and its REST API is supported when the `IdentityReplicationPolicy` is in use, a number of limitations apply, see Streams Replication Manager replication flows and replication policies on page 28.

Streams Replication Manager consists of two main components (roles) that are each responsible for different tasks related to data replication. These are the Streams Replication Manager Driver and Streams Replication Manager Service roles.

The Streams Replication Manager Driver role is based on the Kafka Connect Framework and utilizes Connect connectors and connector tasks to replicate Kafka data. These connectors and tasks produce raw metrics that describe the replication process. The Streams Replication Manager service is responsible for collecting, processing, and exposing these metrics. The metrics can be queried using the Streams Replication Manager Service REST API or viewed in Streams Messaging Manager on the **Replications** page.

Note: For a more in depth look at the internal architecture of Streams Replication Manager as well as how the Driver and Service roles work, see Streams Replication Manager Architecture.

The following sections provide an overview on how raw Streams Replication Manager metrics are collected, how they are processed into metrics that can be used for monitoring.

Raw metric collection

Because the Streams Replication Manager Driver is built on top of the Kafka Connect framework, the connectors and connector tasks of the Driver produce raw Kafka Connect metrics. Although there are many metrics that exist within the Connect framework, Streams Replication Manager only collects the following raw metrics:

- Status metrics are collected from connectors.
- Status, byte-rate, replication-latency-ms, checkpoint-latency-ms, and replication-records-lag metrics are collected from connector tasks.

All of these metrics have two intrinsic properties. They have an origin and a subject. The origin denotes where the metric is collected from. The subject denotes what entity the metric describes. Based on these properties, the raw metrics can be organized into two groups. Connector-originated metrics, which describe the replication flows, and task-originated metrics, which describe the replication of individual topic partitions.

Metrics collected from connectors are connector-originated. Connectors within the framework of Streams Replication Manager represent the replications (also referred to as replication flows). Therefore, the subject of connector-originated metrics are the replications. Because the subject is the replication flow, connector-originated metrics have a wide scope and describe the replication process from a high level. Think of a two cluster replication scenario. The replication flow acts as a bridge between the two clusters. Connector-originated metrics describe the bridge, or in other words, the replication.

Metrics collected from connector tasks are task-originated. Tasks within the framework of Streams Replication Manager represent the actual replication work that is performed. The tasks are the processes that replicate the messages between the source and target clusters from the topic partitions. As a result, the subject of task-originated metrics is the topic partition. Compared to connector-originated metrics, task-originated metrics have a restricted scope and describe the replication process at a low level.

The raw metrics collected by Streams Replication Manager do not provide good insights on the replication process as a whole and cannot be used as is for monitoring. For example, even though connector-originated metrics describe the replications, a replication is always made up of multiple connector instances (or workers), which may even run on different nodes. Therefore, the raw metrics of a single connector instance (or worker) only provide partial data on the replication. Because of this, raw metrics are not exposed by the Streams Replication Manager service. Instead, Streams Replication Manager processes and aggregates them before exposing them for consumption on the REST API.

Metrics processing and aggregation

Evaluating a single metric is rarely useful. For example, looking at the transmitted bytes of an ongoing replication for a single topic partition in the last 30 seconds does not provide good insight on what is actually happening with the topic or the replication. To provide metrics that describe the replication process in a meaningful way, statistics derived from the raw data are required. In other words, the raw metric data must be processed before it can be exposed for consumption. Creating statistics involves gathering data, in this case raw metrics, and deriving their characteristics through processing and aggregation. The result of this are the metrics that Streams Replication Manager exposes on its REST API.

When processing raw data, Streams Replication Manager uses a combination of the following two strategies:

Streams Replication Manager collects and processes metrics over a time period

This is done because having access to the momentary status of a single entity does not provide real value. On the other hand, data gathered over time is descriptive and provides a good filter for temporary or short lived performance issues. Streams Replication Manager gathers data over time using sliding time-window averaging in the case of numeric metrics.

Streams Replication Manager collects and processes metrics across different entities

When monitoring Streams Replication Manager, the entities that you want to monitor are the topics and replications. This is because it is the metrics of the topics and replications that can give you tangible information on the status of Streams Replication Manager. However, there are no raw metrics available that directly describe topics. Additionally, only status metrics are available for the replications; raw metrics measuring byte rate, latency, or lag are not. Nonetheless, Streams Replication Manager can still provide metrics on the topics and replications by way of aggregating available raw metrics across different sets of entities.

Specifically, Streams Replication Manager gathers the raw metrics that describe the topic partitions (the task-originated raw metrics). Next, it aggregates the data over all topic partitions that belong to a topic. That is, over the topic. The resulting data are the metrics that describe the topic.

Metrics that describe the replications are produced in a similar way. Streams Replication Manager gathers the raw partition metrics and aggregates them over a certain replication. To be precise, over all topics that take part in that replication. The resulting data are the metrics that describe the replication.

All resulting metrics reported for the entities include the statistical minimum, maximum, average, and sum of the sample.



Note: Not all statistics are meaningful. For example, the sum of replication speed over all topic partitions does not give good insight on the overall performance of the replication.

Based on how the data is processed, the exposed metrics can be categorized into the following levels:

Cluster-level metrics

Cluster-level metrics are aggregated over a certain time frame and (contrary to the name) over a certain **replication**. These metrics reflect the overall performance of a certain replication.

Topic-level metrics

Topic-level metrics are aggregated over a certain time frame and over a topic. They reflect the overall progress and status of a replication with regards to a single topic.

Available numeric metrics

Table 1: Streams Replication Manager numeric metrics

name	level	description
byte-rate	cluster	The average number of bytes that are being replicated per second, aggregated over a replication
byte-rate	topic	The average number of bytes that are being replicated per second, aggregated over a topic
replication-latency-ms	cluster	Time it takes records to replicate from source to target cluster, aggregated over a replication
replication-latency-ms	topic	Time it takes records to replicate from source to target cluster, aggregated over a topic
checkpoint-latency-ms	cluster	Time it takes consumer group offsets to replicate from source to target cluster, aggregated over a replication
checkpoint-latency-ms	topic	Time it takes consumer group offsets to replicate from source to target cluster, aggregated over a topic
replication-records-lag	cluster	The number of “to be replicated” messages (difference between the source offset of the last replicated message and the log end offset of the source) aggregated over a replication. The “sum” statistics of the metric describes the overall lag of the replication with respect to a replication flow.
replication-records-lag	topic	The number of “to be replicated” messages (difference between the source offset of the last replicated message and the log end offset of the source) aggregated over a topic. The “sum” statistics of the metric describes the overall lag of the replication in a topic.

Notes about calculation of metrics

When monitoring replication and checkpoint latency metrics, keep in mind the following regarding their calculation.

replication-latency-ms

Replication latency is the difference between the record timestamp of the message as saved in the source cluster and the time of the commit in the performing task. Because the source and target clusters in a replication are two distinct clusters, the record timestamp of the message and the commit time will come from two different clusters. As a result, the clocks of the two clusters must be in sync. Otherwise, Streams Replication Manager will calculate the latency incorrectly, which can lead to negative or other misleading values being reported

checkpoint-latency-ms

Checkpoint latency is the difference between the time when the data necessary for creating a checkpoint is fetched (this includes consumer group offsets) and when the checkpoint is eventually committed.

Status metrics

Unlike raw metrics related to latency, record lag, or byte rate, raw status metrics are not numeric. These are simple strings of text that describe the state of the connector or task. For example, connectors will report that they are paused, running, or stopped. Connector tasks will report that they are running, stopped, or failed.

Similarly to other raw metrics, status metrics are not exposed directly by Streams Replication Manager and are also processed and aggregated. Streams Replication Manager uses status metrics as input to create detailed status messages that describe the overall status of a replication and as input for health checks.

The status messages that Streams Replication Manager produces are more complex than the raw metrics used for input. They include the reason and cause of why a specific replication is in the reported state. These messages can be viewed on the **Replications** page of the Streams Messaging Manager UI.

Related Information

Streams Replication Manager Service REST API
Streams Replication Manager Service REST API Reference
Monitoring Kafka cluster replications using Streams Messaging Manager

Streams Replication Manager Service REST API

The Streams Replication Manager Service REST API can be used to monitor your replication workloads. Learn about the endpoints, endpoint versions, and the data that the endpoints expose.

The processed, aggregated, and in part derived metrics produced by the Streams Replication Manager Service are available for consumption through the various endpoints of the Streams Replication Manager Service REST API. Exposed metrics are aggregated over a predefined set of samples and a sliding time window. In addition to aggregated values, the endpoints also return the statistical minimum, maximum, average, and sum of the corresponding sample.

Accessing the API and its endpoints

If you have access to a running instance of Streams Replication Manager, you can find a comprehensive list of the endpoints, their descriptions, as well as example responses for all endpoints on the Streams Replication Manager Service REST API's Swagger UI.

To access the Swagger UI, select the Streams Replication Manager service in Cloudera Manager and go to Web UI SRM Service Swagger UI *[***CLUSTER NAME***]* . Alternatively, you can also view the *Streams Replication Manager REST API Reference* for a comprehensive list of the endpoints.



Tip: If you are accessing the Swagger UI on a running cluster, you can make calls to each endpoint directly on the UI using the Try it out option.

Endpoint versions

REST API endpoints are grouped based on their version. There exists V2 and V1 (legacy) endpoints. Both V1 and V2 endpoints are similar in content and role. Additionally, there is considerable overlap in functionality. However, V1 is provided only for backward compatibility. As a result, Cloudera recommends that you always use V2 endpoints when monitoring Streams Replication Manager. V2 is more sophisticated, offers more endpoints, and allows the target clusters to be specified by the user. The same is not true for V1 endpoints where the target cluster is fixed. It is always the target cluster of the Streams Replication Manager Service instance that serves the REST call.

Internal and non-internal endpoints

In addition to version, the endpoints are also grouped based on whether they are internal or not. Cloudera recommends that you always use non-internal endpoints for monitoring. Internal endpoints are reserved for internal use by Streams Replication Manager. For example, if the Remote Querying feature is set up in your deployment, the Streams Replication Manager Service instances in the deployment use the internal endpoints to query each other's metrics.

Streams Messaging Manager integration

The endpoints made available on the API can also be used by services that integrate with Streams Replication Manager. Streams Messaging Manager, for example, queries the replication metrics using the endpoints and makes the data it collected available for consumption on the **Replications** page of the Streams Messaging Manager UI. In other words, the replication insights that you can view in Streams Messaging Manager are based on the metrics that Streams Replication Manager produces. For more information on how Streams Messaging Manager integrates with Streams Replication Manager as well as the **Replications** page, see *Monitoring Kafka Cluster Replications using Streams Messaging Manager*.

Related Information

Streams Replication Manager Service REST API Reference
Monitoring Kafka cluster replications using Streams Messaging Manager
Streams Replication Manager monitoring and metrics
Streams Replication Manager Service architecture
Remote Querying

Streams Replication Manager replication flows and replication policies

Get familiar with the concepts of replication flows and replication policies. Additionally, learn about the default replication policies shipped with Streams Replication Manager as well as the most commonly used types of replication flows.

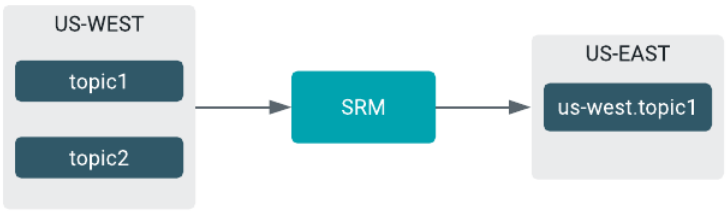
Replication flows

Replication involves sending records and consumer group checkpoints from a source cluster to a target cluster. In Streams Replication Manager, a replication flow (also referred to as a replication or flow) specifies a source and target cluster pair, the direction in which data is flowing and the topics that are being replicated. Source and 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, you must specify which topics to replicate with the srm-control command line tool.

Replication in Streams Replication Manager 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.

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 8: Simple Replication Flow Example



Replication policies

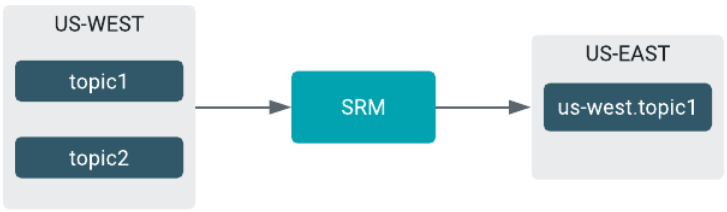
In any replication flow, the selected source topics are replicated to remote (replicated) topics on the target cluster. The basic rules of how these topics are replicated is defined by the replication policy that is used by Streams Replication Manager. Cloudera Runtime includes two replication policies by default. These are the `DefaultReplicationPolicy` and `IdentityReplicationPolicy`. The main difference between the two policies is how they name remote topics. In addition to these policies, Streams Replication Manager also supports the use of custom replication policies.

DefaultReplicationPolicy

The `DefaultReplicationPolicy` is the default and Cloudera-recommended replication policy. This is because the `DefaultReplicationPolicy` is capable of automatically detecting replication loops and supports all monitoring features provided by the Streams Replication Manager Service.

This policy prefixes the remote topic's name with the cluster name (alias) of the source topics. For example, the topic1 topic from the us-west source cluster creates the us-west.topic1 remote topic on the target cluster.

Figure 9: Simple Replication Flow Example



If a remote topic is also replicated, 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 topi c1 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.

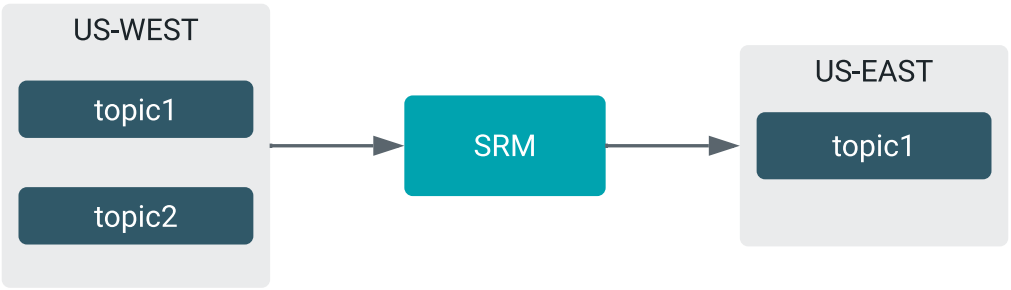
Figure 10: Complex Replication Flow Example



Tip: You might want to have your Kafka consumers read messages from both source and remote topics simultaneously. To achieve this when the `DefaultReplicationPolicy` is in use, Kafka consumers should include a wildcard topic name pattern. For example, suppose that you have a cluster, us-east. This cluster contains a source topic, topic1, and a remote topic from another cluster, us-west.topic1. In such a case, you can use the (.*\.)?topic1 pattern, which matches both topics.

IdentityReplicationPolicy

The `IdentityReplicationPolicy` does not change the names of remote topics. When this policy is in use, topics retain the same name on both source and target clusters. For example, the topic1 topic from the us-west source cluster creates the topic1 remote topic on the target cluster.



This type of replication is also referred to as prefixless replication. This replication policy is recommended for deployments where Streams Replication Manager is used to aggregate data from multiple streaming pipelines. Alternatively, this replication policy can also be used if the deployment requires MirrorMaker1 (MM1) compatible replication.

The `IdentityReplicationPolicy` has the following limitations:

- Replication loop detection is not supported. As a result, you must ensure that topics are **not** replicated in a loop between your source and target clusters.
- The /v2/topic-metrics/{target}/{downstreamTopic}/{metric} endpoint of Streams Replication Manager Service v2 API does not work properly with prefixless replication. Use the /v2/topic-metrics/{source}/{target}/{upstreamTopic}/{metric} endpoint instead.
- The replication metric graphs shown on the **Topic Details** page of the Streams Messaging Manager UI do not work with prefixless replication.
- Reverse checkpointing is not compatible with the `IdentityReplicationPolicy` because the feature relies on cluster prefixes to identify replica topics and correctly transform replica topic names back to their original counterparts.

For more information on how to configure your Streams Replication Manager service to use this policy, see *Enabling prefixless replication*.

Custom replication policies

If neither the `DefaultReplicationPolicy` or `IdentityReplicationPolicy` fit your business requirements, you can develop your own replication policy implementation. Developing and using your own replication policy enables you to gain full control over how Streams Replication Manager replicates data. For more information, see the `ReplicationPolicy` interface in the Kafka Java documentation

Important: Cloudera provides limited support for deployments that use custom replication policies. Additionally, some key features including replication monitoring with the Streams Replication Manager Service will not work if a custom replication policy is in use. Whenever possible, Cloudera recommends that you use the `DefaultReplicationPolicy`.

Bidirectional replication flows

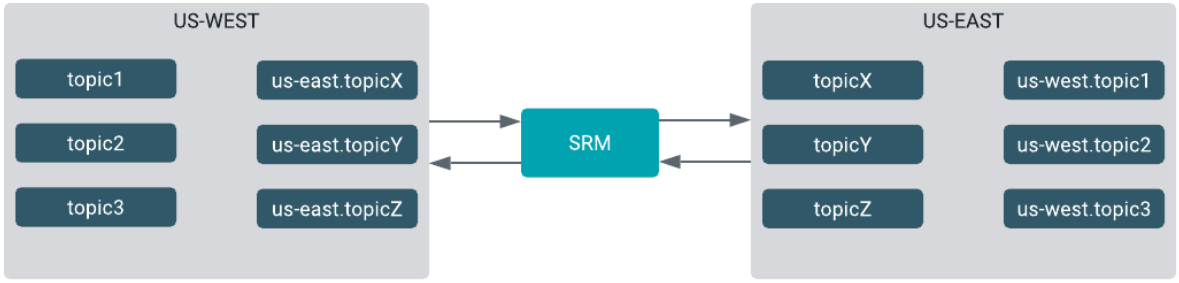
Streams Replication Manager supports bidirectional replication flows. A bidirectional replication flow is a setup where the topics of two or more clusters are mutually replicated. Records sent to one cluster are replicated to the other and the other way around. You can configure any number of clusters in this way.

A common issue with a bidirectional setup is that you can easily create replication loops. A replication loop is when a topic from one cluster is replicated to another and that same topic is replicated back to the source. The `DefaultReplicationPolicy` is capable of detecting replication loops. If this policy is in use, Streams Replication Manager will never replicate records in an infinite loop. As a result, you can freely use regex patterns in your allow and deny lists without having to worry about accidental replication loops.

The `IdentityReplicationPolicy`, on the other hand, cannot detect replication loops. As a result, if using this policy, you must ensure that topic allow and deny lists are correctly set up without any loops in the replication.

For example, consider the following bidirectional setup.

Figure 11: Bidirectional Replication Flow



This replication setup can be configured with the srm-control as follows:

```
srm-control topics --source us-west --target us-east --add "topic.*?"
```

```
srm-control topics --source us-east --target us-west --add "topic.*?"
```

These commands add all topics on both clusters to the replication allowlist using a regex pattern. With the `DefaultReplicationPolicy`, these commands would not create replication loops. This is because the `DefaultReplicationPolicy` automatically detects that the topics with the prefixes do not need to be replicated to the source.

Using these srm-control commands with the `IdentityReplicationPolicy` would, however, result in an endless replication loop. Remote topics would be replicated back to their source. In a case like this, the allow and deny lists must be correctly configured so that no loops are present. For example:

```
srm-control topics --source us-west --target us-east --add topic1, topic2, t
opic3
```

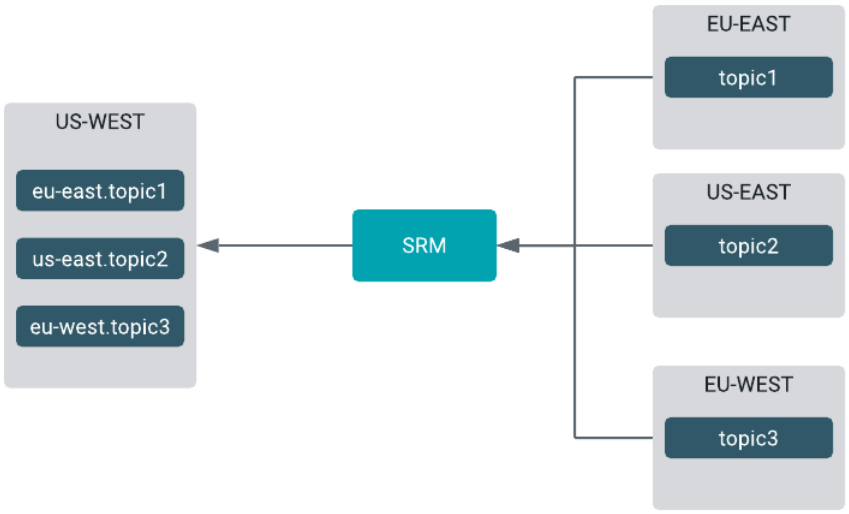
```
srm-control topics --source us-east --target us-west --add topicX, topicY, t
opicZ
```

Note: Regex patterns can still be used with the `IdentityReplicationPolicy`, but you must ensure that the regex pattern matches do not create any loops.

Fan-in and fan-out replication flows

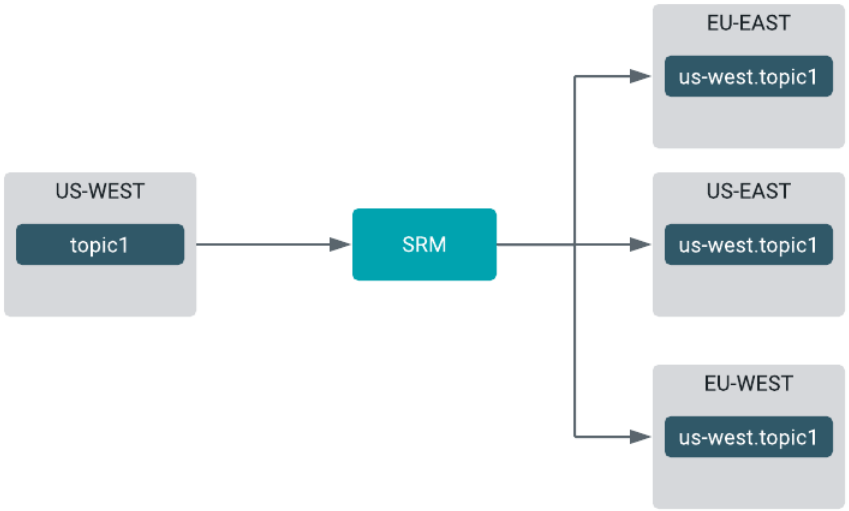
You can construct fan-in replication flows, where records from multiple source clusters are aggregated in a single target cluster.

Figure 12: 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 13: Fan-out Replication Flow



Related Information

Kafka Java documentation - ReplicationPolicy

Streams Replication Manager remote topic discovery

Learn how Streams Replication Manager keeps track of remote topics and how this behavior allows Streams Replication Manager to provide full support for prefixless replication with the IdentityreplicationPolicy.

In order to provide replication monitoring, Streams Replication Manager must discover and keep track of the topics that are replicated. By default, Streams Replication Manager achieves this using an internal Kafka topic, which contains information regarding the currently replicated topics. The behavior is as follows:

1. Streams Replication Manager Drivers periodically check which topics must be replicated.
2. This check results in a list of source – target topic pairs that must be replicated.
3. The Streams Replication Manager Driver writes these source – target pairs into the srm-meta.internal Kafka topic, which is an internal compact topic located in the target cluster.
4. The Streams Replication Manager Service scans srm-meta.internal and consumes the latest message. The latest message contains the currently replicated topics as well as source – target topic name mappings.

Remote topic detection takes into consideration the current state of the replication flow. For example, assume that you have a remote topic, us-east.orders that was replicated at some point in time, but is not currently replicated anymore. Streams Replication Manager correctly identifies that this topic is no longer replicated and will not keep track of it as a remote topic indefinitely. The replication metrics provided by the Streams Replication Manager Service also correctly reflect this state of the topic.

In addition, the use of an internal topic to store replication metadata allows Streams Replication Manager to correctly identify which target maps to which source without relying solely on topic prefixes. This makes it possible for Streams Replication Manager to provide replication monitoring even if a replication policy different than the `DefaultReplicationPolicy` is in use. Most notably, this enables replication monitoring when Streams Replication Manager is configured for prefixless replication with the `IdentityReplicationPolicy`.

Automatic group offset synchronization

Automatic group offset synchronization is a feature in Streams Replication Manager that automates the export and application of translated consumer group offsets. Enabling this feature can simplify the manual steps that you need to take to migrate consumer groups in a failover or failback scenario.

Streams Replication Manager automatically translates consumer group offsets between clusters. While the offset mappings are created by Streams Replication Manager, they are not applied by default to the consumer groups of the target cluster. As a result, by default, migrating consumer groups from one cluster to another involves running the srm-control offsets and kafka-consumer-groups tools. The srm-control offsets tool exports translated offsets, kafka-consumer-groups resets and applies the translated offsets on the target cluster.

You can automate this process by enabling automatic group offset synchronization. If automatic group offset synchronization is enabled, the translated group offsets of the source cluster are automatically exported from the source cluster and are applied on the target cluster (they are written to the __consumer_offsets topic). If you choose to enable this feature, running srm-control offsets and kafka-consumer-groups is not required to migrate consumer groups. You only need to restart and redirect consumers to consume from the new cluster.

Although automatic group offset synchronization can simplify migrating consumer groups, ensure that you understand the following about its behavior:

- Automatic group offset synchronization does not fully automate a failover or failback process. It only allows you to skip certain manual steps. Consumers must be restarted and redirected to the new cluster even if the feature is enabled.
- Offsets are synced at a configured interval. As a result, it is not guaranteed that the latest translated offsets are applied. If you want to have the latest offsets applied, Cloudera recommends that you export and apply consumer group offsets manually. The exact interval depends on sync.group.offsets.interval.seconds and emit.checkpoints .interval.seconds.
- The checkpointing frequency configured for Streams Replication Manager can have an effect on the group offset synchronization frequency. The frequency of group offset synchronization can be configured with sync.gro up.offsets.interval.seconds. However, specifying an interval using this property might not result in the offsets being synchronized at the set frequency. This depends on how emit.checkpoints.interval is configured. The emit .checkpoints.interval property specifies how frequently offset information is fetched (checkpointing). Because offset synchronization can only happen after offset information is available, the frequency configured with the emit.checkpoints.interval property might introduce additional latency. For example, assume that you set offset synchronization to 60 seconds (default), but have checkpointing set to 120 seconds. In such a case, offset synchronization happens every 60 seconds, but because offset information is only refreshed every 120 seconds, in practice offsets are only synchronized every 120 seconds.

- Offsets are only synchronized for the consumers that are inactive in the target cluster. This is done so that the Streams Replication Manager does not override the offsets in the target cluster.
- During a failover-failback scenario, if a consumer group makes progress in the replica topic in the backup cluster and a failback occurs, the progress made in the replica topic is lost, and messages are re-processed from the original topic.

Additionally, Streams Replication Manager checkpointing behavior might affect automatic group offset synchronization in the following ways:

- By default, Streams Replication Manager checkpointing only uses the latest offset sync record, which means that consumer groups behind the latest mapping cannot be checkpointed. If the consumer group is behind by more than the value of offset.lag.max property in replicating the records to the offset sync, the offset translation is skipped because the offset is too far in the past to translate accurately.

  Note: In some cases, Streams Replication Manager checkpointing can also take older synchronized offsets into account.

- Streams Replication Manager checkpointing reads the offset syncs topic to create offset mappings for committed consumer group offsets. In some cases, a mapping might not available in offset syncs. In such a case, Streams Replication Manager copies the source offset, which might not be a valid offset in the replica topic. This can cause issues when auto offset sync is also turned on.
- If sync.group.offsets.enabled is set to true, Kafka only emits checkpoints for partitions that have offset syncs. Partitions for which no offsets are committed (because they cannot be translated) do not appear when describing consumer offsets in the target cluster.

Related Information

Configuring automatic group offset synchronization

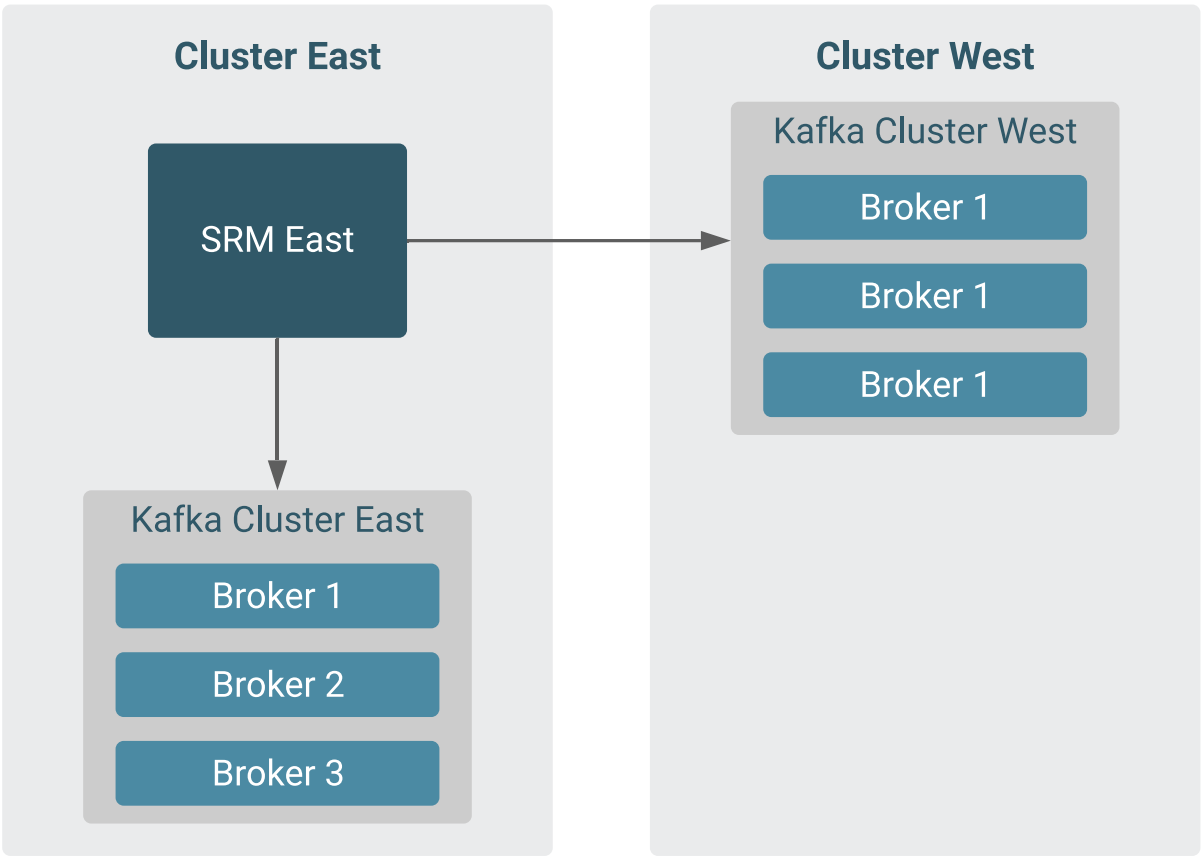
Understanding co-located and external clusters

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

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

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

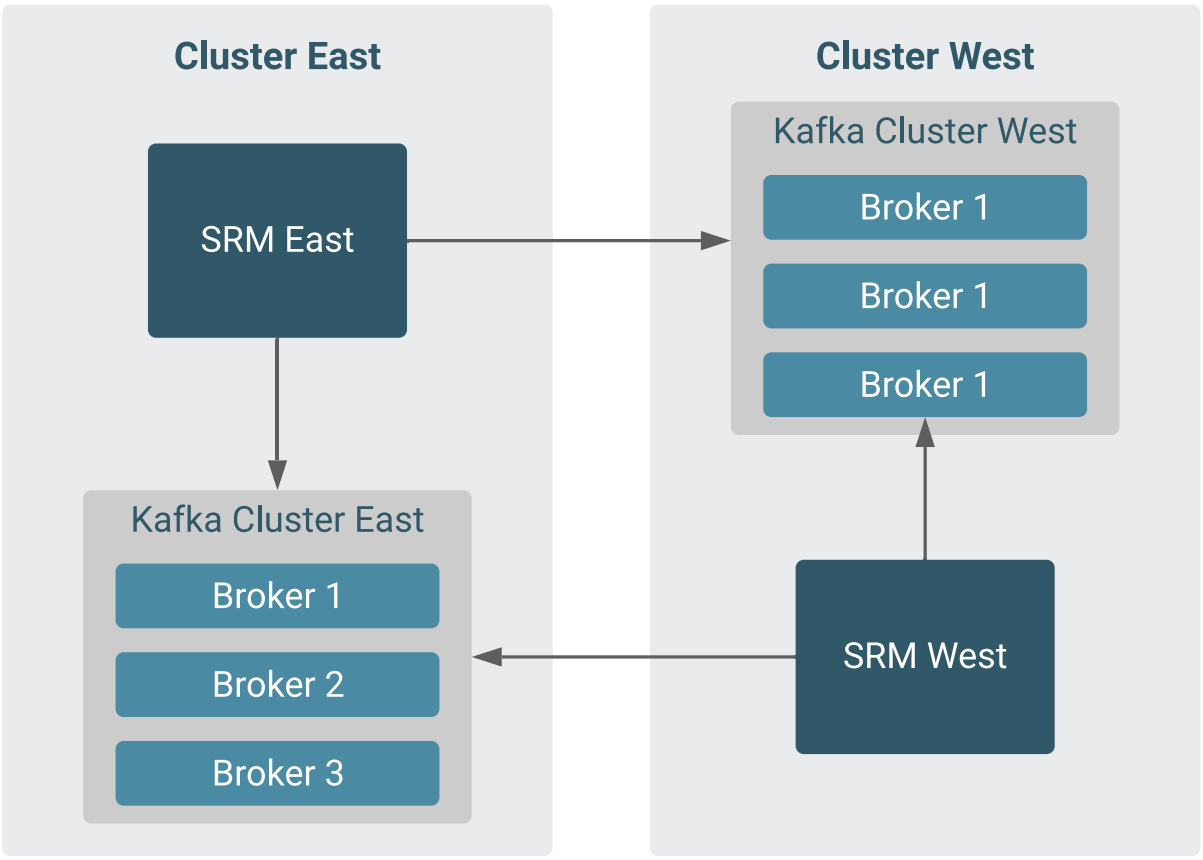
For example, consider the following deployment:



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

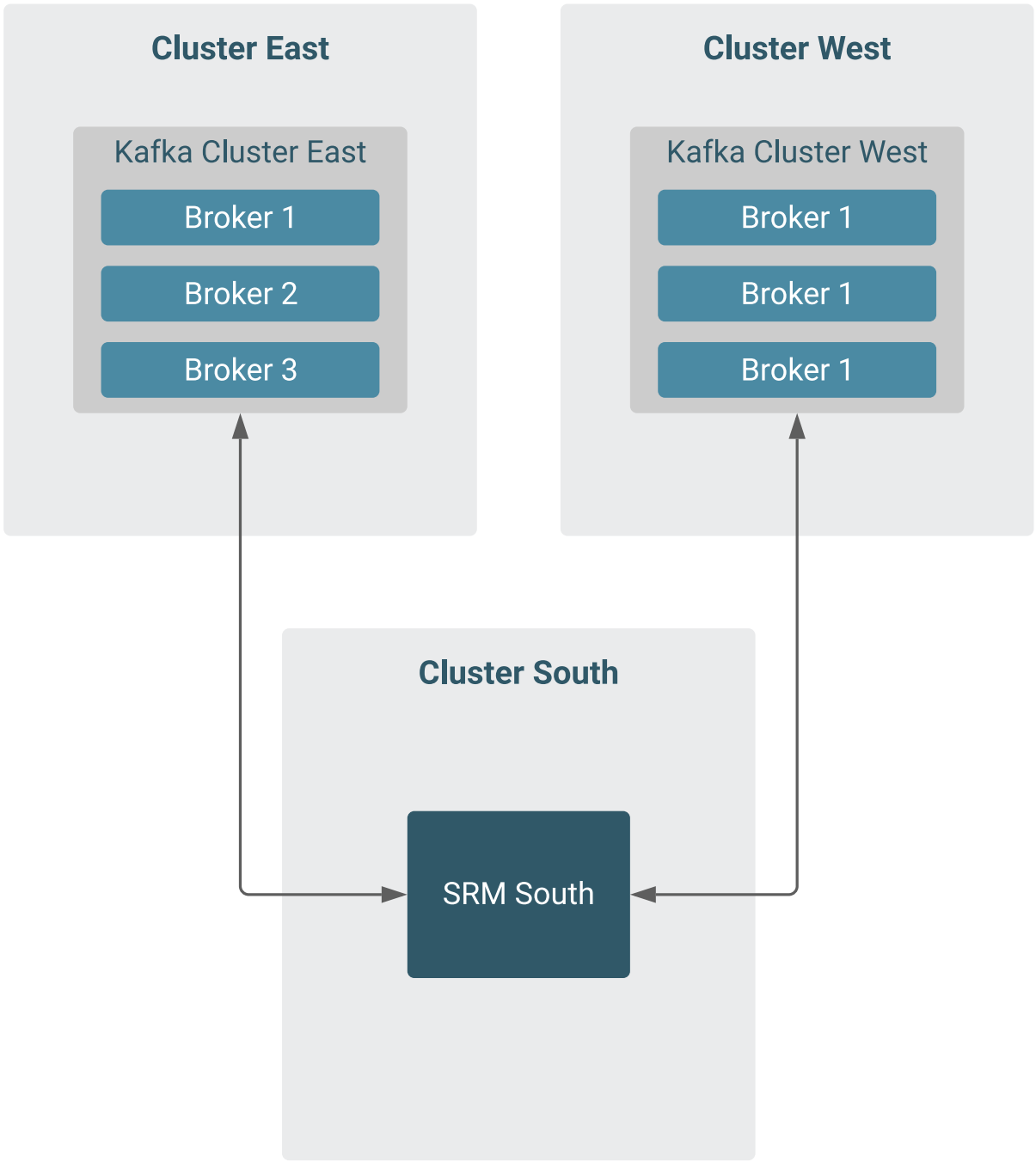
In a more advanced deployment with multiple Streams Replication Manager services, a single Kafka cluster will fall into both categories. From the perspective of a specific Streams Replication Manager service a cluster will be co-located, while for others it will be external.

For example, consider the following deployment:



In this example, both clusters have a Kafka cluster as well as Streams Replication Manager. From the perspective of Streams Replication Manager East, Kafka Cluster East is co-located, Kafka Cluster West is external. From the perspective of Streams Replication Manager 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 Streams Replication Manager. Instead, data is replicated by a Streams Replication Manager instance deployed on a separate cluster. From the perspective of Streams Replication Manager 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 Streams Replication Manager service and the srm-control tool. In general, a co-located Kafka cluster requires less configuration than external Kafka clusters. This is because Cloudera Manageris able to automatically pass certain configuration properties about the co-located Kafka cluster to Streams Replication Manager. External Kafka clusters on the other hand must be fully configured and specified manually.

For more information on how to configure and set up Streams Replication Manager 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 Streams Replication Manager security overview.

Understanding Streams Replication Manager properties, their configuration and hierarchy

There are a number of configuration properties that Streams Replication Manager 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.

Streams Replication Manager 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, Streams Replication Manager 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 Streams Replication Manager. 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
  ```

- Global worker level - prefixed with the workers. prefix

  For example:

  ```
  workers.scheduled.rebalance.max.delay.ms
  ```

- Global connector level - prefixed with the connectors. prefix

  For example:

  ```
  connectors.producer.override.buffer.memory
  ```

- 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 Streams Replication Manager 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. Streams Replication Manager 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 Streams Replication Manager:

Streams Replication Manager service level properties

These configurations are specific to Streams Replication Manager. Additionally, they are applied to Streams Replication Manager 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 Streams Replication Manager Driver to use the same protocol.

Kafka client connection properties

Client connection properties are used by Streams Replication Manager 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 Streams Replication Manager 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 Streams Replication Manager 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 Streams Replication Manager in all target clusters.

Other worker properties that do not use the listed prefixes can still be configured globally. However, they must be configured on a global worker level using the workers. prefix.

Kafka Connect connector properties

These configurations are applied to all connectors in all replications run internally by the Streams Replication Manager Driver. The Connect connector properties supported at this level are as follows:

- errors.retry.timeout
- errors.retry.delay.max.ms
- errors.tolerance
- errors.log.enable
- errors.log.include.messages
- config.properties.exclude
- config.properties.blacklist
- config.property.filter.class
- consumer.poll.timeout.ms
- admin.timeout.ms
- refresh.topics.enabled
- refresh.groups.enabled
- sync.topic.configs.enabled
- sync.topic.acls.enabled
- emit.heartbeats.interval.seconds
- emit.checkpoints.interval.seconds
- sync.group.offsets.enabled
- sync.group.offsets.interval.seconds
- replication.policy.class
- replication.policy.separator
- heartbeats.topic.replication.factor
- checkpoints.topic.replication.factor
- offset-syncs.topic.replication.factor
- offset.lag.max
- offset-syncs.topic.location
- connect.start.task.timeout.ms
- disable.source.topic.auto.creation
- copy.source.offset.in.header.enabled

For example, you can configure sync.group.offsets at this level to sync the translated consumer group offsets.

Other connector properties that are not listed can still be configured globally. However, they must be configured on a global connector level using the connectors. prefix.

Global worker level

Global worker level configuration is achieved by using the workers. prefix. This level supports Kafka Connect worker properties only. Properties specified with the workers. prefix are applied to all Connect workers run internally by the Streams Replication Manager Driver.

The difference between the global worker level and the global level (no prefix) is that on a global level only a handful of Connect worker properties are supported. The global worker level on the other hand supports all Connect worker properties.

For example, scheduled.rebalance.max.delay.ms is not supported on a global level, it is not applied to all Connect workers if it is set without a prefix. However, specifying the property on a global worker level is supported. That is, when the property is set with the workers. prefix, it is applied to all workers.

Global connector level

Global connector level configuration can be achieved by prefixing the configuration property with the connectors. prefix. This level supports Kafka Connect connector properties only. Properties that use this prefix are applied to all Connect connectors run internally by the Streams Replication Manager Driver.

The difference between the global connector level and the global level (no prefix) is that on a global level only a handful of Connect connector properties are supported. The global connector level on the other hand supports all Connect connector properties.

The global connector level prefix can also be used to tweak all consumers and producers that are used by the Streams Replication Manager Driver. For example, by using the connectors.consumer. prefix, any consumer property can be specified to tweak the consumers used in the replications. Similarly, the connectors.producer.override. prefix can be used to override producer configurations in the 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 Streams Replication Manager:

Kafka client connection properties

These configurations are used by Streams Replication Manager to connect to the Kafka cluster identified by the alias. Streams Replication Manager is capable of generating connection configurations securely for most use-cases and Kafka clusters either through automation, or with Kafka Credentials (Administration External Accounts Kafka 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 Streams Replication Manager, 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 Streams Replication Manager:

Kafka 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.

Kafka Connect 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

Reverse Checkpointing

The reverse checkpointing feature in Streams Replication Manager enables the tracking and replication of consumer offsets from a target cluster back to a source cluster. This ensures that progress made by consumer groups on a backup cluster is preserved and translated back to the primary cluster during a failback scenario.

Failback in standard replication scenarios

In standard replication scenarios, checkpointing supports the replication of committed offsets from an upstream (source) topic to a downstream (replica) topic. This allows consumers to fail over to a backup cluster and resume consumption without scanning the entire topic. However, without reverse checkpointing, the downstream-to-upstream direction is not supported. When a consumer group fails back to the original source cluster after processing data on the backup, the offsets committed on the backup are not translated back to the source.

This results in one of the following scenarios upon failback:

Missing offsets:

If no offset is recorded for the consumer group in the source cluster, the consumer resets to the earliest or latest offset, potentially leading to massive data duplication or skipped data. This can occur when the consumer group only started consuming from the backup cluster after the failover and never consumed from the source cluster.

Obsolete offsets:

The consumer resumes from an old offset recorded before the failover, which means that data that was consumed on the backup cluster is consumed again, leading to data duplication.

Reverse Checkpointing logic

Reverse checkpointing addresses the failback data duplication issue through offset translation in the downstream-to-upstream direction. To perform this translation, Streams Replication Manager needs bidirectional replication to be enabled. Then, in case of failback, the replication flow from the backup to the source can use the offset-syncs generated by the original source-to-backup flow as well as the reverse flow. This allows Streams Replication Manager to map the offsets from the replica topic back to the equivalent offsets in the source topic, and so minimize message duplication.

Limitations

Reverse checkpointing aims to minimize the volume of duplicate messages upon failback. However, it does not eliminate duplicates completely, which means that consumers need to be prepared to handle duplicates, even if you

are using exactly-once schematic in Kafka and Streams Replication Manager. Also, the exact order of messages is not guaranteed to be preserved in every failover and failback scenario.