Streams Replication Manager for HDF and HDP 1.0.0

# Configuring SRM

**Date published: 2019-09-20**
**Date modified: 2019-09-20**

**https://docs.cloudera.com/**

## Legal Notice

# Contents

# Configuration Overview

Learn how to configure SRM with the `srm.properties` configuration file.

SRM configuration is managed through a single configuration file. The default location of the configuration file for .rpm based installs is `/opt/streams-replication-manager/config`. For ZIP or TAR based installations the default location is `$SRM_CONF_DIR/srm.properties`. The default name of the file is `srm.properties`.

Command line tools provided with SRM read configuration properties from the configuration file. All tools automatically collect configuration information from the default file. If required it is possible to store the configuration file at a different location. In this case however, you need to specify the configuration file and its location when using the tools. Specifying an alternate location for the file is achieved with the `--config` option which is supported by all SRM command line tools.

The `srm.properties` file accepts all Kafka client properties available in the version of Kafka that you are using. Additionally, it supports a number of SRM specific configuration properties. For a comprehensive list of SRM specific properties, see Configuration Properties Reference.

Configuration properties for SRM can be set on three different levels, which are the following:

- Top level: Top level or global configuration is achieved by adding the property on its own. For example:

  ```
  replication.factor=3
  ```

- Cluster level: Cluster level configuration can be achieved by prepeding the configuration property with a cluster alias. For example:

  ```
  primary.replication.factor=3
  ```

- Replication level: Replication level configuration can be achieved by prepeding the configuration property with the name of the replication. For example:

  ```
  primary->secondary.replication.factor=3
  ```

## Minimum Configuration

At minimum the configuration file has to contain cluster aliases, cluster connection information, and at least one `cluster->cluster` replication that is enabled.

Cluster aliases are specified with the `clusters` property. Aliases are arbitrary names defined by the user. They are used in other configuration properties as well as with the SRM command line tools to refer to the clusters added for replication. For example:

```
#Kafka cluster aliases
clusters = primary, backup
```

Each cluster alias has to have connection information associated with it. Connection information is specified with the `bootstrap.servers` property. You add connection information by prepending the `bootstrap.servers` property with a cluster alias and adding the address of the Kafka broker as the value. When configuring connection information, add each cluster to a new line. If a cluster has multiple hosts, add them to the same line but delimit them with commas. For example:

```
#Kafka broker addresses
primary.bootstrap.servers = primary-cluster1.vpc.example.com:9092, primary-c
luster2.vpc.example.com:9092, ...
backup.bootstrap.servers = backup-cluster1.vpc.example.com:9092, backup-cl
uster1.vpc.example.com:9092, ...
```

Cluster replications can be set up and enabled as follows:

```
primary->backup.enabled = true
backup->primary.enabled = true
```

The default `srm.properties` shipped with SRM contains examples for cluster aliases, connection information, and `cluster->cluster` replications. In addition, it also contains a number of other pre-set properties. These properties however are not required for SRM to function, they are included to give users a basic example of how the configuration file can be set up. Cloudera recommends that you study the example configuration provided in Configuration Examples to find out more about how SRM should be set up for different replication scenarios.

**Related Information**

Configuration Examples

Configuration Properties Reference

# New Topic and Consumer Group Discovery

Understand how SRM discovers new topics and consumer groups on source clusters.

The discovery and replication of newly created topics or consumer groups is not instantaneous. SRM checks source clusters for new topics and consumer groups periodically, as controlled by the `refresh.topics.interval.seconds` and `refresh.groups.interval.seconds` properties. By default both properties are set to 10 minutes. As a result, the discovery and replication of new topics or groups can take up to 10 minutes.

Cloudera does not recommend using a refresh interval lower than the default value for production environments as it can lead to severe performance degradation.

# Configuration Examples

Configuration examples for typical SRM architectures.

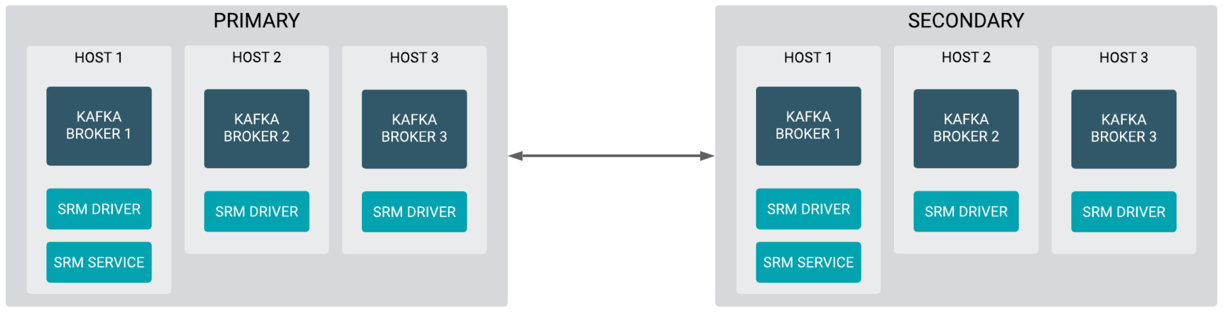
## Bidirectional Replication of Active Clusters

Configuration example for two active Kafka clusters setup with bidirectional replication.

### About this task

A typical scenario involves two active Kafka clusters within the same region but in separate availability zones. Clients can connect to either cluster in case one is temporarily unavailable. This example demonstrates the steps required to set up a deployment with two clusters configured with bidirectional replication. Additionally, it also provides example commands to start replication between clusters.

**Figure 1: Bidirectional Replication of Active Clusters**



## Procedure

1. Install the following configuration file on every broker.

```
# two clusters: primary and secondary
clusters = primary, secondary
primary.bootstrap.servers = primary_host1:9092, primary_host2:9092, prim
ary_host3:9092
secondary.bootstrap.servers = secondary_host1:9092, secondary_host2:9092,
 secondary_host3:9092

# bidirectional replication between primary and secondary
primary->secondary.enabled = true
secondary->primary.enabled = true
```

2. Run the `srm-driver` with the `--clusters` option on every Kafka broker host.

   Running the `srm-driver` with the `--clusters` option allows you to specify which clusters each instance of the driver should write to. In this example, both instances of the driver are set up to read data from all clusters but only write to the cluster they are running on. This allows you to distribute replication workloads.

   - On `primary` hosts:

   ```
   srm-driver --clusters primary
   ```

   - On `secondary`hosts:

   ```
   srm-driver --clusters secondary
   ```

3. Optional: If you want to monitor replication, run `srm-service` on one host per cluster. Each instance of the service should target the cluster its running on.

```
# on a single primary host
srm-service --target primary
# on single secondary host
srm-service --target secondary
```

**4.** Replicate data between clusters with the following commands:

```
srm-control topics --source primary --target secondary --add ".*"
```

```
srm-control topics --source secondary --target primary --add ".*"
```

## Cross Data Center Replication of Multiple Clusters

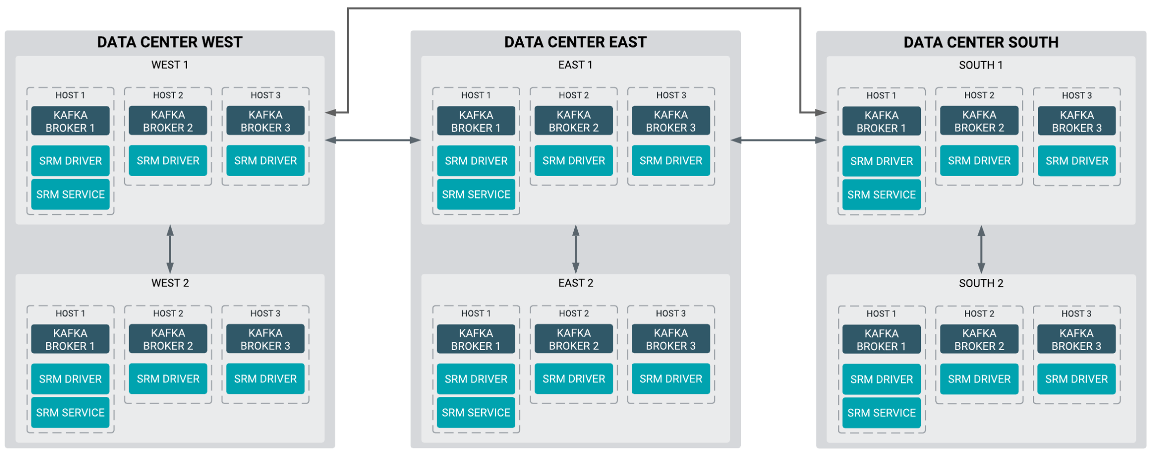
Configuration example with three data centers that have two Kafka clusters each.

### About this task

In more advanced deployments, you may have multiple Kafka clusters in each of several data centers. To prevent creating a fully-connected mesh of all Kafka clusters, Cloudera recommends leveraging a single Kafka cluster in each data center for cross data center replication.

This example demonstrates the steps required to configure a deployment with three data centers that each have two Kafka clusters. Additionally, it also provides example commands to start bidirectional replication of all topics within each data center and an example on how to replicate a single topic across all data centers.

**Figure 2: Cross Data Center Replication of Multiple Clusters**



### Procedure

**1.** Install the following configuration file on every broker.

```
# six clusters in different data centers
clusters = west1, west2, east1, east2, south1, south2
# clusters in west DC
west1.bootstrap.servers = west1_host1:9092, west1_host2:9092, west1_host3
:9092
west2.bootstrap.servers = west2_host1:9092, west2_host2:9092, west2_host3:
9092

# clusters in east DC
east1.boostrap.servers = east1_host1:9092, east1_host2:9092, east1_host3
:9092
east2.boostrap.servers = east2_host1:9092, east2_host2:9092, east2_host3:9
092
# clusters in south DC
```

```
south1.boostrap.servers = south1_host1:9092, south1_host2:9092, south1_h
ost3:9092
south2.boostrap.servers = south2_host1:9092, south2_host2:9092, south2_hos
t3:9092
# cross data center replication via west1, east1, south1
west1->east1.enabled = true
west1->south1.enabled = true
east1->west1.enabled = true
east1->south1.enabled = true
south1->west1.enabled = true
south1->east1.enabled = true

# bidirectional replication within each DC
west1->west2.enabled = true
west2->west1.enabled = true
east1->east2.enabled = true
east2->east1.enabled = true
south1->south2.enabled = true
south2->south1.enabled = true
```

2. Run the `srm-driver` with the `--clusters` option on every Kafka broker host.

   Running the `srm-driver` with the `--clusters` option allows you to specify which clusters each instance of the driver should write to. In this example, both instances of the driver are set up to read data from all clusters but only write to the cluster they are running on. This allows you to distribute replication workloads.

   - In the west data center:

     ```
     srm-driver --clusters west1 west2
     ```
   - In the east data center:

     ```
     srm-driver --clusters east1 east2
     ```
   - In the south data center:

     ```
     srm-driver --clusters south1 south2
     ```

3. Optional: If you want to monitor replication, run `srm-service` on one host per cluster. Each instance of the service should target the cluster its running on.

```
# on a single host in west1:
$ srm-service --target west1

# on a single  host in west2:
$ srm-service --target west2

# on a single host in  east1:
$ srm-service --target east1

# on a single host in east2:
$ srm-service --target east2

# on a single host in south1:
$ srm-service --target south1

# on a single host in south2:
$ srm-service --target south2
```

4. Replicate topics between hosts within each data center:

```
srm-control topics --source west1 --target west2 --add ".*"
```

```
srm-control topics --source west2 --target west1 --add ".*"
```

```
srm-control topics --source east1 --target east2 --add ".*"
```

```
srm-control topics --source east2 --target east1 --add ".*"
```

```
srm-control topics --source south1 --target south2 --add ".*"
```

```
srm-control topics --source south2 --target south1 --add ".*"
```

5. Replicate `topic1` across all data centers:

```
srm-control topics --source west1 --target east1 --add topic1,west2.topic1
```

```
srm-control topics --source west1 --target south1 --add topic1,west2.top
ic1
```

```
srm-control topics --source east1 --target west1 --add topic1,east2.topic1
```

```
srm-control topics --source east1 --target south1 --add topic1,east2.top
ic1
```

```
srm-control topics --source south1 --target west1 --add topic1,south2.to
pic1
```

```
srm-control topics --source south1 --target east1 --add topic1,south2.to
pic1
```

# Configuration Properties Reference

A collection of all SRM specific configuration properties.

### Table 1:

| Property | Default Value | Description |
|---|---|---|
| sync.topic.configs.enabled | true | Enables the monitoring of the source cluster for configuration changes. |
| sync.topic.acls.enabled | true | Enables the monitoring of the source cluster for ACL changes. |
| emit.heartbeats.enabled | true | Enables periodic emission of heartbeats. |
| emit.heartbeats.interval.seconds | 5 (seconds) | The interval at which SRM emits heartbeats. |
| emit.checkpoints.enabled | true | Enables periodic emission of consumer offset information. |
| emit.checkpoints.interval.seconds | 60 (seconds) | The interval at which SRM emits checkpoint information. |

| Property | Default Value | Description |
|---|---|---|
| refresh.topics.enabled | true | Enables a periodical check for new topics on source clusters. |
| refresh.topics.interval.seconds | 10 (minutes) | The interval at which SRM looks for new topics on source clusters. |
| refresh.groups.enabled | true | Enables a periodical check for new consumer groups on source clusters. |
| refresh.groups.interval.seconds | 10 (minutes) | The interval at which SRM looks for new consumer groups on source clusters. |
| replication.policy.class | org.apache.kafka.connect.mirror.DefaultReplicationPolicy | Replication policies to use. Use `LegacyReplicationPolicy` to mimic legacy MirrorMaker behavior. |
| heartbeats.topic.replication.factor | 3 | Replication factor used for internal heartbeat topics. |
| checkpoints.topic.replication.factor | 3 | Replication factor used for internal checkpoints topics. |
| offset-syncs.topic.replication.factor | 3 | Replication factor used for internal offset-syncs topics. |