

Configuring Apache Kafka

Date published: 2019-12-18

Date modified: 2022-08-30



Legal Notice

© Cloudera Inc. 2024. All rights reserved.

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

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

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

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

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

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

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

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

Contents

Operating system requirements.....	4
Performance considerations.....	4
Quotas.....	5
Configuring quotas.....	6
JBOD.....	7
JBOD setup.....	8
JBOD Disk migration.....	9
Setting user limits for Kafka.....	11
Rolling restart checks.....	11
Configuring rolling restart checks.....	13
Configuring the client configuration used for rolling restart checks.....	14
Cluster discovery with multiple Apache Kafka clusters.....	15
Cluster discovery using DNS records.....	15
A records and round robin DNS.....	16
client.dns.lookup property options for client.....	16
CNAME records configuration.....	17
Connection to the cluster with configured DNS aliases.....	18
Cluster discovery using load balancers.....	18
Setup for SASL with Kerberos.....	20
Setup for TLS/SSL encryption.....	22
Connecting to the Kafka cluster using load balancer.....	22
Configuring Kafka ZooKeeper chroot.....	23
Kafka rack awareness.....	23
Rack awareness for Kafka brokers.....	24
Configuring rack awareness for Kafka brokers.....	24
Rack awareness for Kafka consumers.....	25
Configuring rack awareness for Kafka consumers.....	26
Rack awareness for Kafka producers.....	27
Configuring rack awareness for Kafka producers.....	28

Operating system requirements

A collection of operating system requirements for Kafka.

SUSE Linux Enterprise Server (SLES)

Unlike CentOS, SLES limits virtual memory by default. Changing this default requires adding the following entries to the `/etc/security/limits.conf` file:

```
* hard as unlimited
* soft as unlimited
```

Kernel Limits

There are three settings you must configure properly for the kernel.

File Descriptors

You can set file descriptors in Cloudera Manager by going to `KafkaConfigurationMaximumProcessFileDescriptors` and setting the required value. Cloudera recommends a configuration of 100000 or higher.

Max Memory Map

You must configure the maximum number of memory maps in your specific kernel settings. Cloudera recommends a configuration of 32000 or higher.

Max Socket Buffer Size

Set the buffer size larger than any Kafka send buffers that you define.

Performance considerations

A collection of basic recommendations for Kafka clusters.

The simplest recommendation for running Kafka with maximum performance is to have dedicated hosts for the Kafka brokers and a dedicated ZooKeeper cluster for the Kafka cluster. If that is not an option, consider these additional guidelines for resource sharing with the Kafka cluster:

Running in VMs

It is common practice in modern data centers to run processes in virtual machines. This generally allows for better sharing of resources. Kafka is sufficiently sensitive to I/O throughput that VMs interfere with the regular operation of brokers. For this reason, it is generally not recommended to run Kafka in VMs. However, if you are running Kafka in a virtual environment you will need to rely on your VM vendor for help with optimizing Kafka performance.

Do not run other processes with Brokers or ZooKeeper

Due to I/O contention with other processes, it is generally recommended to avoid running other such processes on the same hosts as Kafka brokers.

Keep the Kafka-ZooKeeper Connection Stable

Kafka relies heavily on having a stable ZooKeeper connection. Putting an unreliable network between Kafka and ZooKeeper will appear as if ZooKeeper is offline to Kafka. Examples of unreliable networks include:

- Do not put Kafka/ZooKeeper nodes on separated networks
- Do not put Kafka/ZooKeeper nodes on the same network with other high network loads

Quotas

Learn about Apache Kafka quotas, quota types, client groups, quota violation enforcement and detection, quota storage, as well as how you can configure quotas.

Kafka can enforce quotas on produce and fetch requests. Producers and consumers can use very high volumes of data. This can monopolize broker resources, cause network saturation, and generally deny service to other clients and the brokers themselves. Quotas protect against these issues and are important for large, multitenant clusters where a small set of clients using high volumes of data can degrade the user experience.

Quota types

There are two types of client quotas that can be enforced by Kafka brokers for each group of clients sharing a quota. These are as follows.

Network bandwidth quotas (byte-rate thresholds)

Network bandwidth quotas are defined as the byte rate threshold for each group of clients sharing a quota. Each group of clients can publish or fetch a maximum of X bytes/second per broker before clients are throttled.

Request rate quotas (CPU utilization thresholds)

Request rate quotas are defined as the percentage of time a client can utilize on request handler I/O threads and network threads of each broker within a quota window. A quota of n% represents n% of one thread, so the quota is out of a total capacity of $((\text{num.io.threads} + \text{num.network.threads}) * 100)\%$.

Each group of clients can use a total percentage of up to n% across all I/O and network threads in a quota window before being throttled. Since the number of threads allocated for I/O and network threads are typically based on the number of cores available on the broker host, request rate quotas represent the total percentage of CPU that each group of clients sharing the quota can use.

Client Groups

The identity of Kafka clients is the user principal, which represents an authenticated user in a secure cluster. In a cluster that supports unauthenticated clients, the user principal is a grouping of unauthenticated users chosen by the broker using a configurable `PrincipalBuilder`.

A client-id logically identifies an application making a request. A single client-id can span multiple producer and consumer instances. The tuple (user + client-id) defines a secure logical group of clients that share both user principal and client-id.

You can apply quotas to tuple (user + client-id), user, or client-id groups. For a given connection, the most specific quota matching the connection is applied. All connections of a quota group share the quota configured for the group. For example, if the tuple (user="test-user", client-id="test-client") has a produce quota of 10 MB/sec, then that quota is shared across all producer instances using the test-user user with the test-client client-id.

Quota violation detection

To detect quota violations quickly, byte-rate and thread utilization are measured over multiple small windows. For example, 30 windows that are each 1 second long. Having large measurement windows, for example, 10 windows that are 30 seconds each, leads to large bursts of traffic followed by long delays.

Quota violation enforcement

When a client exceeds its specified quota, the broker attempts to throttle the client instead of returning an error message. Throttling happens using the following logic.

1. The broker calculates the amount of delay needed to bring a client under its quota.

2. The broker sends a response to the client that includes the delay that the broker calculated. If the broker is responding to a fetch request, the response only contains the delay. The data that was requested is not included in the response.
3. The broker mutes the channel to the client. Any additional requests that the broker receives from the client are only processed after the delay is over.
4. If a client receives a response that includes a delay, the client refrains from sending further requests to the broker. This means that requests from a throttled client are blocked by both broker and client.

Quota storage and precedence

Quota configurations are stored in ZooKeeper. Specifically, user and tuple (user + client-id) quota configurations are written to ZooKeeper under /config/users, and client-id quota configurations are written under /config/clients.

The order of precedence for quota configuration is as follows.

1. /config/users/[***USER**]/clients/[***CLIENT ID**]
2. /config/users/[***USER**]/clients/<default>
3. /config/users/[***USER**]
4. /config/users/<default>/clients/[***CLIENT ID**]
5. /config/users/<default>/clients/<default>
6. /config/users/<default>
7. /config/clients/[***CLIENT ID**]
8. /config/clients/<default>

Configuring quotas

Learn how to configure Apache Kafka quotas.

By default, each client receives an unlimited quota. However, customizing quotas is possible. You can define quota configuration on the level of tuple (user + client-id), user, and client-id groups. In addition, there are two categories of quota configuration, default and custom.

For example, you can define a default quota configuration that applies to all clients, but also specify custom configurations that only apply to a specific subset of clients. Both default and custom configurations can be specified for any of the levels. Additionally, changes to quota configuration are dynamic and are effective immediately. This means that a broker restart is not required for the configuration to take effect.

Configuration is done with the kafka-configs tool using the --alter, --add-config, --entity-type, --entity-name, and --entity-default options.

The following collects various example commands that configure quotas as well as an example demonstrating how you can describe quotas.

Custom quota configuration examples

- Configure a custom quota for a tuple (user + client-id)

```
kafka-configs --bootstrap-server [***HOST***]:[***PORT***] --alter --add-config 'producer_byte_rate=1024,consumer_byte_rate=2048,request_percentage=200' --entity-type users --entity-name [***USER NAME***] --entity-type clients --entity-name [***CLIENT-ID***]
```

- Configure a custom quota for a user

```
kafka-configs --bootstrap-server [***HOST***]:[***PORT***] --alter --add-config 'producer_byte_rate=1024,consumer_byte_rate=2048,request_percentage=200' --entity-type users --entity-name [***USER NAME***]
```

- Configure a custom quota for a client-id

```
kafka-configs --bootstrap-server [***HOST***]:[***PORT***] --alter --add-config 'producer_byte_rate=1024,consumer_byte_rate=2048,request_percentage=200' --entity-type clients --entity-name [***CLIENT-ID***]
```

Default quota configuration examples

Notice that default quota configurations are set by specifying the `--entity-default` option instead of `--entity-name`.

- Configure a default client-id quota for a user

```
kafka-configs --bootstrap-server [***HOST***]:[***PORT***] --alter --add-config 'producer_byte_rate=1024,consumer_byte_rate=2048,request_percentage=200' --entity-type users --entity-name [***USER NAME***] --entity-type clients --entity-default
```

- Configure a default quota for a user

```
kafka-configs --bootstrap-server [***HOST***]:[***PORT***] --alter --add-config 'producer_byte_rate=1024,consumer_byte_rate=2048,request_percentage=200' --entity-type users --entity-default
```

- Configure a default quota for a client-id

```
kafka-configs --bootstrap-server [***HOST***]:[***PORT***] --alter --add-config 'producer_byte_rate=1024,consumer_byte_rate=2048,request_percentage=200' --entity-type clients --entity-default
```

Describing quotas

In addition to configuring quotas using `kafka-configs`, you can also describe the quota configuration for any specific tuple (user + client-id), user, or client-id group. This is done by using the `--describe` option. For example, the quota configuration of a specific tuple can be retrieved with the following command.

```
kafka-configs --bootstrap-server [***HOST***]:[***PORT***] --describe --entity-type users --entity-name [***USER NAME***] --entity-type clients --entity-name [***CLIENT-ID***]
```

JBOD

Overview on Kafka with JBOD.

JBOD refers to a system configuration where disks are used independently rather than organizing them into redundant arrays (RAID). Using RAID usually results in more reliable hard disk configurations even if the individual disks are not reliable. RAID setups like these are common in large scale big data environments built on top of commodity hardware. RAID enabled configurations are more expensive and more complicated to set up. In a large number of environments, JBOD configurations are preferred for the following reasons:

- **Reduced storage cost:** RAID-10 is recommended to protect against disk failures. However, scaling RAID-10 configurations can become excessively expensive. Storing the data redundantly on each node means that storage space requirements have to be multiplied because the data is also replicated across nodes.
- **Improved performance:** Just like HDFS, the slowest disk in RAID-10 configuration limits overall throughput. Writes need to go through a RAID controller. On the other hand, when using JBOD, IO performance is increased as a result of isolated writes across disks without a controller.

JBOD setup

Learn how to set up JBOD in your Kafka environment.

Before you begin

Consider the following before using JBOD support in Kafka:

- Manual operation and administration: Monitoring offline directories and JBOD related metrics is done through Cloudera Manager. However, identifying failed disks and rebalancing partitions between disks is done manually.
- Manual load balancing between disks: Unlike with RAID-10, JBOD does not automatically distribute data across disks. The process is fully manual.

To provide robust JBOD support in Kafka, changes in the Kafka protocol have been made. When performing an upgrade to a new version of Kafka, make sure that you follow the recommended rolling upgrade process.

For more information regarding the JBOD related Kafka protocol changes, see KIP-112 and KIP-113.

Procedure

1. Mount the required number of disks on your system.
2. In Cloudera Manager, set up log directories for all Kafka brokers:
 - a) Go to the Kafka service, select Instances and select the broker.
 - b) Go to Configuration and find the Data Directories property.
 - c) Modify the path of the log directories so that they correspond with the newly mounted disks.



Note: Depending on your setup you may need to add or remove multiple data directories.

- d) Enter a Reason for change, and then click Save Changes to commit the changes.
3. Go to the Kafka service and select Configuration.
 4. Find and configure the following properties depending on your system and use case.
 - Number of I/O Threads
 - Number of Network Threads
 - Number of Replica Fetchers
 - Minimum Number of Replicas in ISR
 5. Set replication factor to at least 3.



Important: If you set replication factor to less than 3, your data will be at risk. In addition, in case of a disk failure, disk maintenance cannot be carried out without system downtime.

6. Restart the service:
 - a) Return to the home page by clicking the Cloudera Manager logo.
 - b) Go to the Kafka service and select Actions Rolling Restart
 - c) Check the Restart roles with stale configurations only checkbox and click Rolling restart.
 - d) Click Close when the restart has finished.

Results

JBOD disks are set up in your Kafka environment.

Related Information

[KIP-112](#)

[KIP-113](#)

JBOD Disk migration

Learn how to migrate existing Kafka partitions to JBOD configured disks.

About this task

Migrating data from one disk to another is achieved with the `kafka-reassign-partitions` tool. The following instructions focus on migrating existing Kafka partitions to JBOD configured disks.



Note: Cloudera recommends that you minimize the volume of replica changes per command instance. Instead of moving 10 replicas with a single command, move two at a time in order to save cluster resources.

Before you begin

- Set up JBOD in your Kafka environment. For more information, see [JBOD Setup](#).
- Collect the log directory paths on the JBOD disks where you want to migrate existing data.
- Collect the broker IDs of the brokers you want to migrate data to.
- Collect the name of the topics you want to migrate partitions from.



Note: Output examples in these instructions are cleaned and formatted to make them easily readable.

Procedure

1. Create a topics-to-move JSON file that specifies the topics you want to reassign. Use the following format:
Use the following format:

```
{ "topics": [ { "topic": "mytopic1" },
               { "topic": "mytopic2" } ],
  "version": 1
}
```

2. Generate the content for the reassignment configuration JSON with the following command:

```
kafka-reassign-partitions --zookeeper hostname:port --topics-to-move-json-file topics to move.json --broker-list broker 1, broker 2 --generate
```

Running the command lists the distribution of partition replicas on your current brokers followed by a proposed partition reassignment configuration.

Example output:

```
Current partition replica assignment
{ "version": 1,
  "partitions": [
    { "topic": "mytopic2", "partition": 1, "replicas": [2, 3], "log_dirs": [ "any", "any" ] },
    { "topic": "mytopic1", "partition": 0, "replicas": [1, 2], "log_dirs": [ "any", "any" ] },
    { "topic": "mytopic2", "partition": 0, "replicas": [1, 2], "log_dirs": [ "any", "any" ] },
    { "topic": "mytopic1", "partition": 2, "replicas": [3, 1], "log_dirs": [ "any", "any" ] },
    { "topic": "mytopic1", "partition": 1, "replicas": [2, 3], "log_dirs": [ "any", "any" ] }
  ]
}

Proposed partition reassignment configuration
```

```
{
  "version": 1,
  "partitions": [
    {
      "topic": "mytopic1",
      "partition": 0,
      "replicas": [4, 5],
      "log_dirs": ["any", "any"]
    },
    {
      "topic": "mytopic1",
      "partition": 2,
      "replicas": [4, 5],
      "log_dirs": ["any", "any"]
    },
    {
      "topic": "mytopic2",
      "partition": 1,
      "replicas": [4, 5],
      "log_dirs": ["any", "any"]
    },
    {
      "topic": "mytopic1",
      "partition": 1,
      "replicas": [5, 4],
      "log_dirs": ["any", "any"]
    },
    {
      "topic": "mytopic2",
      "partition": 0,
      "replicas": [5, 4],
      "log_dirs": ["any", "any"]
    }
  ]
}
```

In this example, the tool proposed a configuration which reassigns existing partitions on broker 1, 2, and 3 to brokers 4 and 5.

3. Copy and paste the proposed partition reassignment configuration into an empty JSON file.
4. Modify the suggested reassignment configuration.

When migrating data you have two choices. You can move partitions to a different log directory on the same broker, or move it to a different log directory on another broker.

- a. 1. To reassign partitions between log directories on the same broker, change the appropriate any entry to an absolute path. For example:

```
{
  "topic": "mytopic1",
  "partition": 0,
  "replicas": [4, 5],
  "log_dirs": ["/JBOD-disk/directory1", "any"]
}
```

2. To reassign partitions between log directories across different brokers, change the broker ID specified in replicas and the appropriate any entry to an absolute path. For example:

```
{
  "topic": "mytopic1",
  "partition": 0,
  "replicas": [6, 5],
  "log_dirs": ["/JBOD-disk/directory1", "any"]
}
```

5. Save the file.
6. Start the redistribution process with the following command:

```
kafka-reassign-partitions --zookeeper hostname:port --reassignment-json-file reassignment_configuration.json --bootstrap-server hostname:port --execute
```



Important: The bootstrap server has to be specified with the `--bootstrap-server` option if an absolute log directory path is specified for a replica in the reassignment configuration JSON file.

The tool prints a list containing the original replica assignment and a message that reassignment has started. Example output:

Current partition replica assignment

```
{
  "version": 1,
  "partitions": [
    {
      "topic": "mytopic2",
      "partition": 1,
      "replicas": [2, 3],
      "log_dirs": ["any", "any"]
    },
    {
      "topic": "mytopic1",
      "partition": 0,
      "replicas": [1, 2],
      "log_dirs": ["any", "any"]
    },
    {
      "topic": "mytopic2",
      "partition": 0,
      "replicas": [1, 2],
      "log_dirs": ["any", "any"]
    },
    {
      "topic": "mytopic1",
      "partition": 2,
      "replicas": [3, 1],
      "log_dirs": ["any", "any"]
    },
    {
      "topic": "mytopic1",
      "partition": 1,
      "replicas": [2, 3],
      "log_dirs": ["any", "any"]
    }
  ]
}
```

```
}
```

Save this to use as the `--reassignment-json-file` option during rollback
Successfully started reassignment of partitions.

7. Verify the status of the reassignment with the following command:

```
kafka-reassign-partitions --zookeeper hostname:port --reassignment-json-file reassignment_configuration.json --bootstrap-server hostname:port --verify
```

The tool prints the reassignment status of all partitions. Example output:

```
Status of partition reassignment:
Reassignment of partition mytopic2-1 completed successfully
Reassignment of partition mytopic1-0 completed successfully
Reassignment of partition mytopic2-0 completed successfully
Reassignment of partition mytopic1-2 completed successfully
Reassignment of partition mytopic1-1 completed successfully
```

Results

Existing Kafka partitions are migrated to JBOD configured disks.

Related Information

[JBOD setup](#)

[kafka-reassign-partitions](#)

Setting user limits for Kafka

Learn more about Kafka User limits and how to monitor them.

Kafka opens many files at the same time. The default setting of 1024 for the maximum number of open files on most Unix-like systems is insufficient. Any significant load can result in failures and cause error messages such as `java.io.IOException...(Too many open files)` to be logged in the Kafka or HDFS log files. You might also notice errors such as this:

```
ERROR Error in acceptor (kafka.network.Acceptor)
java.io.IOException: Too many open files
```

Cloudera recommends setting the value to a relatively high starting point, such as 32,768.

You can monitor the number of file descriptors in use on the Kafka Broker dashboard. In Cloudera Manager:

1. Go to the Kafka service.
2. Select a Kafka Broker.
3. Open [Charts Library Process Resources](#) and scroll down to the File Descriptors chart.

Rolling restart checks

You can configure Cloudera Manager to perform a check on Kafka brokers during a rolling restart. Using this check can ensure that Kafka brokers stay healthy after the rolling restart. There are multiple types of checks available, each providing a different level of guarantee on Kafka broker and cluster health.

By default, during a rolling restart, Cloudera Manager only checks whether restarting a Kafka broker has failed or succeeded. As a result of this behaviour, Kafka brokers might go into a state where some of the topics and partitions become unreachable by the clients. For example, by default Cloudera Manager might restart a broker while the

previous broker is not fully ready for operation. This can cause outages and corrupted log indexes. To avoid such issues, you can configure Cloudera Manager to perform a more thorough check on the Kafka brokers during a rolling restart.

Check types and configuration

There are multiple checks available, each providing a different (higher) level of guarantee on Kafka cluster and broker health. The type of check performed is configured with the Cluster Health Guarantee During Rolling Restart property. The property has four different settings, each setting corresponds to a different type of check. The available settings and the check types that the settings correspond to are as follows:

none (default)

This setting disables rolling restart checks. If this option is selected, no checks are performed and no health guarantees are provided.

ready for request

This setting ensures that when a broker is restarted, the restarted broker is accepting and responding to requests made on its service port. The next broker is only restarted after the previous broker is ready for requests.

healthy partitions stay healthy

This setting ensures that no partitions go into an under-min-isr state when a broker is stopped. This is achieved by waiting before each broker is stopped so that all other brokers can catch up with all replicas that are in an at-min-isr state. Additionally, this setting ensures that the restarted broker is accepting and is responding to requests made on its service port before restarting the next broker. This setting ignores partitions which are already in an under-min-isr state.

all partitions stay healthy (recommended)

This setting ensures that no partitions are in an under-min-isr or at-min-isr state when a broker is stopped. This is achieved by waiting before each broker is stopped so that all other brokers can catch up with all replicas that are in an at-min-isr or under-min-isr state. Additionally, this setting ensures that the restarted broker is accepting requests on its service port before the next broker is restarted.

In addition to configuring and enabling these checks using Cluster Health Guarantee During Rolling Restart, a number of other configuration properties are also available that enable you to fine-tune the behaviour of the checks. For detailed steps on how to enable and configure rolling restart checks, see *Configuring rolling restart checks*.

How checks are performed

When Cloudera Manager executes a rolling restart check, it uses the kafka-topics tool to gather information about the brokers, topics, and partitions. The kafka-topics tool requires a valid client configuration file to run. In the case of rolling restart checks, two configuration files are required. One for the kafka-topics commands that are initiated before a broker is stopped, and a separate one for the commands initiated after a broker is restarted. Cloudera Manager automatically generates these client configuration files based on the configuration of the Kafka service. These files can also be manually updated using advanced security snippets.

Using these files, Cloudera Manager executes kafka-topics commands on the brokers. Based on the response from the tool, Cloudera Manager either waits for a specified amount of time or continues with the rolling restart.

Depending on what type of check is configured, Cloudera Manager polls information with kafka-topics at different points in time. As a result, the checks can be categorised in two groups. Pre-checks and post-checks. If either healthy partitions stay healthy or all partitions stay healthy is selected, information is polled both before a broker is stopped (pre-check) and after a broker is restarted (post-check). If the ready for request setting is selected, information is only polled after a broker is restarted.

If a pre-check fails to find a proper state when a broker can be stopped, the check will stop the entire rolling restart process. This can happen if the broker that is about to be stopped still has at-min-isr or under-min-isr partitions after the configured timeout interval is reached. Post-checks behave in a similar way. If the post-check fails to receive validation (a correct exit code) within the specified timeout interval from the kafka-topics command that the broker

is ready for requests, the check will stop the entire rolling restart process. In both of these cases the brokers are not stopped or restarted. The rolling restart fails and the brokers continue to run.



Note: Configuring and using any type of check increases the time required for a rolling restart. This is the result of Cloudera Manager waiting between restarting the brokers. The timeout intervals however can be configured to a lower value if the rolling restart check takes too much time to finish. Alternatively, if you are experiencing timeout related rolling restart failures, you can also configure the timeout intervals to a higher value.

Advanced configuration

There are two scenarios when additional configuration is required. These scenarios are as follows:

Kafka brokers are configured to use a custom listeners

If you configured your Kafka brokers with advanced configuration snippets to use custom listeners (for example a custom host:port pair), you must manually update both client configuration files that Cloudera Manager generates. This is required because Cloudera Manager might not be able to automatically extract the information required to establish a connection with the Kafka brokers when custom listeners are configured. For more information, see *Configuring the client configuration used for rolling restart checks*.

A broker connectivity change is made after rolling restart checks are enabled

A broker connectivity change is any type of change made to listeners, bootstrap servers, ports, or security. If a change like this is made after rolling restart checks are enabled, Cloudera Manager uses the newly set configuration to generate the client configuration files. However, until a restart is executed, the Kafka brokers still operate with the old configuration. As a result, Cloudera Manager will run the kafka-topics tool with an invalid configuration causing the check and the rolling restart to fail. In a case like this, you must disable rolling restart checks until the Kafka brokers are restarted at least once. This can be done by setting Cluster Health Guarantee During Rolling Restart to none. Following the initial restart, the brokers will operate with the new configuration and you can re-enable rolling restart checks.

Configuring rolling restart checks

You can configure Cloudera Manager to perform different types of checks on Kafka brokers during a rolling restart. The type of check performed by Cloudera Manager is configured with the Cluster Health Guarantee During Rolling Restart property. The property has multiple settings, each setting corresponds to a different type of check.

About this task

The following steps walk you through the basic configuration method of how you can enable and configure rolling restart checks.

Before you begin

If your Kafka service is configured to use custom listeners, complete [Configuring the client configuration used for rolling restart checks](#) before continuing with this task.

Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to Configuration.
3. Find and configure the Cluster Health Guarantee During Rolling Restart property.

Select one of the available options. Click the ? icon next to the property name to reveal a full description of each option and the check to which they correspond.

Cloudera recommends that you set this property to all partitions stay healthy to avoid service outages

4. Fine-tune rolling restart check behaviour by configuring the following properties:

- Maximum Allowed Runtime For Kafka Broker Rolling Restart Check
- Retry Interval For Kafka Broker Rolling Restart Check
- Default API Timeout For Kafka Topics Client Used In Kafka Broker Rolling Restart Check

These properties allow you to configure different interval and timeout values related to the rolling restart check. Configure these properties based on your cluster and requirements.

5. Click Save Changes.

6. Restart the Kafka service.

Results

Rolling restart checks are configured and enabled. During any subsequent rolling restarts, Cloudera Manager executes the type of check you configured.

What to do next

If you make any configuration changes related to broker connectivity (security, listeners, port, bootstrap) after rolling restart checks are enabled, you must disable rolling restart checks for the first restart after the change was made. Otherwise, the check and the rolling restart might fail. Following the initial restart, you can re-enable rolling restart checks.

Configuring the client configuration used for rolling restart checks

Cloudera Manager requires Kafka client configuration files to perform rolling restart checks. These files are generated automatically. However, if your Kafka service has custom listeners configured, you must manually update these client configuration files. Otherwise, the rolling restart check might fail.

About this task

When Cloudera Manager executes a rolling restart check, it uses the kafka-topics tool to gather information about the brokers, topics, and partitions. The kafka-topics tool requires a valid client configuration file to run. Cloudera Manager automatically generates two configuration files for this purpose. One is used for the kafka-topics commands initiated before the brokers are stopped, the other, after brokers are restarted.

If your Kafka service is configured to use custom listeners, you must manually update the configuration files generated by Cloudera Manager. This is required because Cloudera Manager might not be able to automatically extract the information required to establish a connection with the Kafka service when custom listeners are configured. The client configuration files can be updated using advanced security snippets.

Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to Configuration.
3. Manually update the client configuration files used during rolling restart checks.

This can be done by adding a valid client configuration to the following advanced configuration snippets:

- Kafka Broker Advanced Configuration Snippet (Safety Valve) for `rolling_restart_check_before_stop_admin_client_configs.properties`
- Kafka Broker Advanced Configuration Snippet (Safety Valve) for `rolling_restart_check_after_start_admin_client_configs.properties`

Ensure that you add the same client configuration to both snippets. The client configuration you add must contain all properties that are required to establish a connection with the brokers. The client configuration you add here is similar to any other client configuration you create for Kafka command line tools. However, this specific

configuration accepts the `bootstrap.servers` property. Use this property to specify your custom host:port pairs that you use as your custom listeners.

The following client configuration example is for a Kafka service that has both TLS/SSL and Kerberos enabled. You can use this example as a template and make changes as needed. For more client configuration examples, see the Securing Apache Kafka publication in the *Streams Messaging* documentation.

```
bootstrap.servers=[***HOST***]:[***PORT***]
security.protocol=SASL_SSL
ssl.client.auth=none
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true storeKey=true keyTab="***PATH TO KEYTAB***" principal=
"***KERBEROS PRINCIPAL***";
ssl.keystore.location=[***PATH TO KEYSTORE.JKS***]
ssl.key.password=[***PASSWORD***]
ssl.keystore.password=[***PASSWORD***]
ssl.keystore.type=jks
ssl.truststore.location=[***PATH TO TRUSTSTORE.JKS***]
ssl.truststore.type=jks
ssl.truststore.password=[***PASSWORD***]
```

4. Click Save Changes.

Results

The client configuration files used by Cloudera Manager during rolling restart checks are configured.

What to do next

Enable and configure rolling restart checks. Complete *Configuring rolling restart checks*.

Related Information

[Streams Messaging](#)

[Configuring rolling restart checks](#)

Cluster discovery with multiple Apache Kafka clusters

When you have multiple Kafka clusters for load balancing or failover purposes, a client can find a suitable cluster either using a DNS server or using a load balancer.

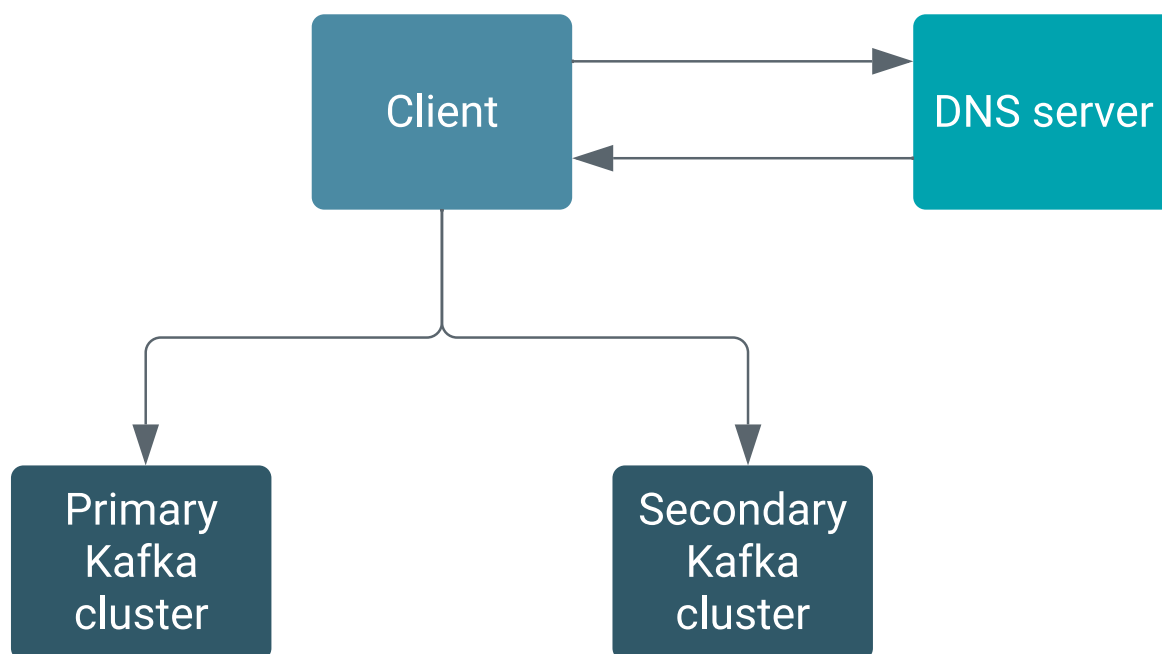
A common architectural pattern for Kafka is to have redundant Kafka clusters across multiple data centers so that applications can either load balance or fail over to the remote data center in case the local one becomes unavailable. Finding a suitable cluster by the client is called cluster discovery. If you connect to a cluster, you do not need all the brokers, one broker is enough. (However, to avoid a single point of failure, it is better to have multiple brokers.) This means that your discovery service should provide at least one active broker. The client gets metadata of all the cluster members after connecting to the broker it got from the service/mechanism. Getting brokers from the discovery service is only useful for the bootstrap phase of the connection. The client must connect to all the brokers directly to operate properly.

The two options for cluster discovery in case of multiple Apache Kafka clusters are using DNS records or load balancers. Cluster discovery using DNS records is the simpler solution.

Cluster discovery using DNS records

Learn about using a DNS server, which is the simplest cluster discovery method.

Kafka has built-in support for cluster discovery using a DNS server as a discovery service. In this case, the client uses a DNS server to resolve hostname aliases and then according to the response, it connects directly to a broker:



A possible solution is shown in the following sections using some DNS records with some examples. The examples are given using a possible syntax for a BIND DNS server, but any other DNS server can be used for the solution. BIND is used here for simplicity.

A records and round robin DNS

Learn about A records and round robin DNS.

A DNS A record is an entry that holds a hostname and the corresponding IP address. You can have multiple A records for different IP addresses using the same hostname as follows (the brokers also have their own FQDNs):

```

; PRIMARY CLUSTER
first.primary.kafka.fqdn.      IN  A      1.2.3.4
second.primary.kafka.fqdn.     IN  A      4.5.6.7
third.primary.kafka.fqdn.      IN  A      6.7.8.9

; DNS alias
primary.kafka.cluster.hostname. IN  A      1.2.3.4
primary.kafka.cluster.hostname. IN  A      4.5.6.7
primary.kafka.cluster.hostname. IN  A      6.7.8.9
  
```

If you try to resolve `primary.kafka.cluster.hostname` multiple times with any tools (`nslookup`, `dig`), you will get different results each time. This is called round robin load balancing, which is used by the DNS server automatically.

The `/etc/hosts` file holds the IP address and hostname mappings (similar to A records in DNS servers), but it is not capable of balancing. You always get the same IP if you try to resolve the same hostname.

In production environments, a low TTL might be used for DNS records for the clients to detect changes as early as possible.

client.dns.lookup property options for client

If you are using DNS aliases, you need to configure the right value for the `client.dns.lookup` property for your setup.

The `client.dns.lookup` property is needed when DNS aliases are used. There can be problems with any security protocol if the `client.dns.lookup` property is not set properly.

The `client.dns.lookup` property can take the following values:

- `client.dns.lookup=default`

In this case, the client connects to the first broker it gets from the DNS response, even if that broker is down. If you only provided a single DNS alias as bootstrap server for the client, the client might be blocked because the mentioned hostname is resolved for multiple brokers where the first broker (to which the client connected) is actually stopped.

- `client.dns.lookup=use_all_dns_ips`

This setup takes care of retrying all the possible IP addresses behind a hostname if the first does not succeed. This setup is also good when brokers have multiple IP addresses and it is possible that those addresses change (especially in Kubernetes environments). This configuration does not handle the case when the hostname used by the client is not a real FQDN for a host. To avoid man-in-the-middle attacks, the client gets SSL handshake exceptions when using SSL and SASL authentication exceptions when using SASL. Since Kafka version 2.6, this is the default.

- `client.dns.lookup=resolve_canonical_bootstrap_servers_only`

This setup ensures a behavior similar to `use_all_dns_ips`, but it also handles the SSL and SASL problem. With this configuration, the client resolves the DNS alias into a list of brokers it maps to and after the bootstrap phase, this behaves the same as `use_all_dns_ips`.



Tip: Cloudera recommends that you set the `resolve_canonical_bootstrap_servers_only` value because this option provides the most fault tolerance.

CNAME records configuration

When you use Kafka's built-in support for cluster discovery you can use CNAME records as shorter alternatives to the longer hostname that is an alias.

In addition to A records, it is also possible to have CNAME records in the DNS servers. It is good to have a simpler/shorter hostname as an alternative for the one that maps to all the brokers.

```
;CNAME record
active.kafka.      IN CNAME      primary.kafka.cluster.hostname.
```

In this case, clients can use the simpler and shorter alternative for the hostname as follows:

```
kafka-topics --list --bootstrap-server active.kafka:9092
```

Using a CNAME also provides the possibility to easily switch between standby and active clusters. Assume there is also a standby cluster defined:

```
; PRIMARY CLUSTER
first.primary.kafka.fqdn.      IN  A      1.2.3.4
second.primary.kafka.fqdn.     IN  A      4.5.6.7
third.primary.kafka.fqdn.      IN  A      6.7.8.9

; DNS alias
primary.kafka.cluster.hostname. IN  A      1.2.3.4
primary.kafka.cluster.hostname. IN  A      4.5.6.7
primary.kafka.cluster.hostname. IN  A      6.7.8.9

; STANDBY CLUSTER
first.standby.kafka.fqdn.      IN  A      4.3.2.1
second.standby.kafka.fqdn.     IN  A      7.6.5.4
third.standby.kafka.fqdn.      IN  A      9.8.7.6

; DNS alias
```

```
standby.kafka.cluster.hostname.    IN    A      4.3.2.1
standby.kafka.cluster.hostname.    IN    A      7.6.5.4
standby.kafka.cluster.hostname.    IN    A      9.8.7.6

;CNAME record
active.kafka.      IN CNAME      primary.kafka.cluster.hostname.
```

If the primary cluster completely stops, it is only needed to change the active.kafka CNAME to point to standby.kafka a.cluster.hostname instead of the primary one:

```
active.kafka.      IN CNAME      standby.kafka.cluster.hostname.
```



Important: Even though using of CNAME records is convenient, they can double the number of DNS queries.

Connection to the cluster with configured DNS aliases

If you only want to use a single hostname for the whole cluster, configure a DNS alias.

This A record setup is a convenient solution for having a single hostname for the whole cluster. You only need to provide a hostname that serves as an alias for the brokers:

```
kafka-topics --list --bootstrap-server primary.kafka.cluster.hostname:9092
```

When connecting your clients to the brokers, the DNS alias is specified and then translated to the actual hostname.

However, a port number must also be specified, and the port you specify is used for all brokers identified by the DNS alias. As a result of this, you must ensure that all brokers identified by a specific DNS alias use the same port number. Otherwise, clients fail to connect even if DNS resolution is successful.

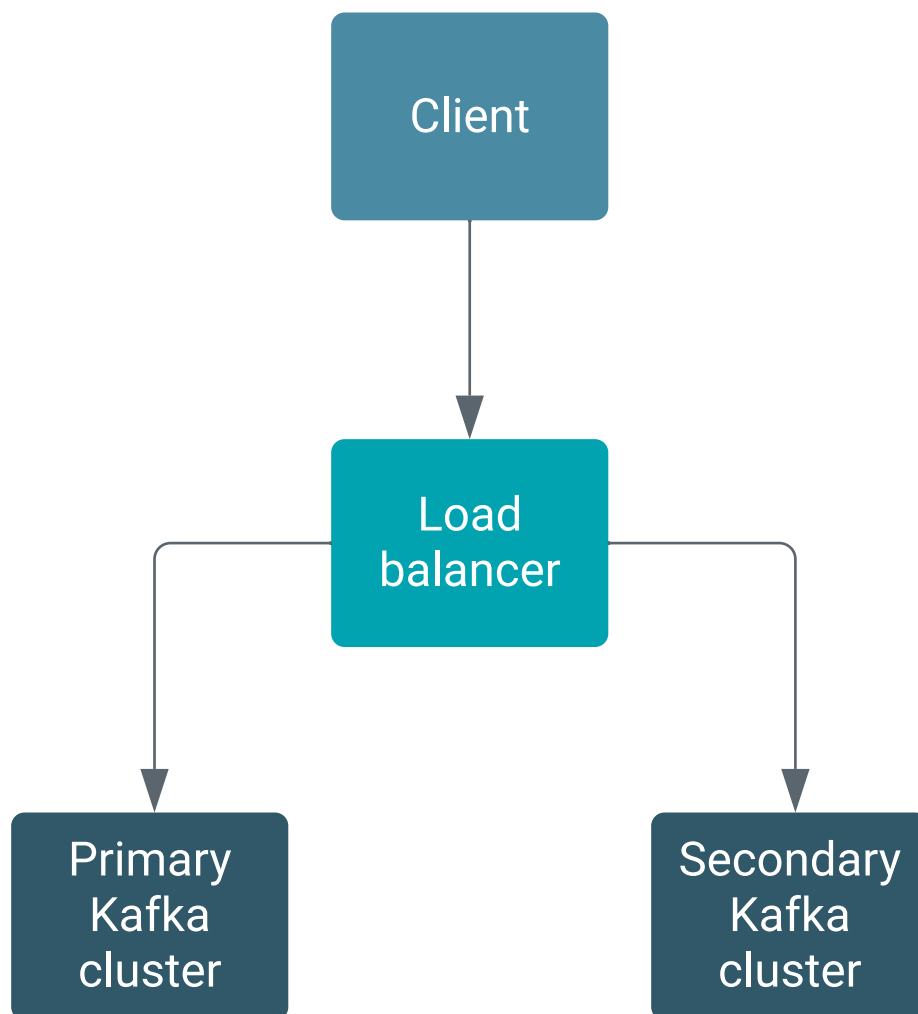
In case of a broker failure, you can change the records in the DNS server to point to other brokers. The next time a client tries to connect, it resolves to a different cluster's broker. This way you do not hardcode broker FQDNs to the bootstrap server list.

If the client has discovered a cluster and is actively using it, and the cluster suddenly stops, the client tries to connect to the bootstrap servers it got from the DNS server before the bootstrap phase. So it does not automatically ask the DNS server for new bootstrap servers and does not fail over to the other cluster, even if the DNS records have already been changed.

Cluster discovery using load balancers

In case of more complex Kafka cluster setups, you might need a cluster discovery solution more sophisticated than a DNS server. In such cases, you should consider using a load balancer.

Cluster discovery using load balancers is less lightweight than using DNS servers, but a viable solution for more complex cases. With a load balancer, it is possible to poll the nodes, check their health status and exclude stopped nodes from targets, automatically redirect requests to living nodes. When using a load balancer the requests are forwarded to a broker as shown in the figure:



Because the client connects to the load balancer and is then forwarded to a broker, SSL handshake and SASL authentication errors can occur (this is a defending mechanism to avoid man-in-the-middle attacks), therefore, additional setup is needed.

The configuration depends on your security protocol:

- No security in the cluster (security protocol is PLAINTEXT)

Setup steps are not required before connecting to the Kafka cluster, the load balancers should work out of the box with Kafka.

- SASL with Kerberos enabled

Perform the setup described in section *Setup for SASL with Kerberos*.

- TLS/SSL encryption is enabled

Perform the setup described in section *Setup for TLS/SSL encryption*.

- SASL with Kerberos and TLS/SSL are both enabled

Perform the setup described in *Setup for SASL with Kerberos* and *Setup for TLS/SSL encryption*.

Related Information

[Setup for SASL with Kerberos](#)

[Setup for TLS/SSL encryption](#)

Setup for SASL with Kerberos

If you are using SASL with Kerberos for authentication, you must configure the load balancer and select the relevant architecture in Cloudera Manager.

Setting the Kafka Broker Load Balancer Host property

Learn how to configure the Kafka Broker Load Balancer Host property to avoid a ticket mismatch when using SASL with Kerberos.

About this task

If you are using SASL with Kerberos in the system and your clients connect to a load balancer that forwards requests to the actual brokers, the client gets a Kerberos service ticket including the load balancer host. This happens because the client believes that the load balancer does not forward requests but it is a broker. Meanwhile, the Kafka brokers in the configured listeners are logged in to Kerberos using their service principal that contains their corresponding FQDN. This results in a ticket mismatch after the client is routed to an actual broker with the load balancer related service ticket.

If you are using SASL_SSL or SASL_PLAINTEXT security protocol in the cluster with Kerberos, you have to set the load balancer's host in Cloudera Manager for the Kafka broker roles by setting the Kafka Broker Load Balancer Host property.

Before you begin

Ensure that you have reviewed [Kerberos-related architecture options](#).

Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to the Configuration tab.
3. Find and configure the Kafka Broker Load Balancer Host property.
Add the hostname of the load balancer.
4. Find and configure the Kafka Broker Load Balancer Listener Port property.



Important: Set the load balancer to forward requests to this port.

5. Click Save Changes.
6. Restart the Kafka service.

Results

After restarting the brokers, the following is automatically set:

- A new load balancer principal is added for each broker's kafka.keytab:

```
kafka_principal/load_balancer_host@REALM
```

By default, the Kafka principal in the cluster is kafka.

- A new listener named LB is added to the kafka.properties:

```
listeners=LB://HOST:9094
```

The LB listener port can be configured by setting the Kafka Broker Load Balancer Listener Port property.

- Similarly to listeners, a new advertised listener is added to the kafka.properties:

```
advertised.listeners=LB://HOST:9093
```

The LB advertised listener advertises the normal Kafka listener's port that is also automatically set (9092 or 9093 depending on SSL settings).

- The security protocol for the new LB listener is added to its corresponding map configuration with the same security protocol as set for the normal listener the clients connect to without using the load balancer:

```
listener.security.protocol.map=LB:SASL_SSL
```

- The JAAS configuration for the LB listener is set to log in with the load balancer principal using the actual Kafka process folder for the keytab path:

```
listener.name.lb.gssapi.sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule
    required doNotPrompt=true useKeyTab=true storeKey=true
    keyTab="/process_folder/kafka.keytab"
    principal="kafka_principal/load_balancer_host@REALM"; "
```



Note: If you have already set any of these properties using advanced configuration snippets (safety valves), the feature does not overwrite your changes. It might simply not work, so you have to extend your advanced configuration snippets manually with the mentioned settings.

- LB can be used with uppercase letters if the load balancer listener should be referenced in property values.
- lb can be used with lowercase letters if the load balancer listener should be referenced in property keys.

Turning off the load balancer listener

If you are no longer using the load balancer, you need to turn off the load balancer listener and restart the Kafka service.

About this task

If the load balancer is not used anymore in front of Kafka brokers, it is enough to clear the Kafka Broker Load Balancer Host property to make it empty and restart the Kafka service. The Kafka broker will not use the load balancer listener port anymore with its listener.

Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to the Configuration tab.
3. Find and clear the Kafka Broker Load Balancer Host property.
4. Click Save Changes.
5. Restart the Kafka service.

Kerberos-related architecture options for Kafka broker behind load balancer setup

Learn about possible Kerberos-related architectures before setting up Kafka brokers behind a load balancer to use SASL with Kerberos.

There are possible Kerberos-related architectures with their own tradeoffs. It is important to consider them to design the system according to needs before setting up Kafka brokers behind a load balancer. There can be differences between the number of Cloudera Manager and Key Distribution Center (KDC) instances connected with the Kafka clusters.

- Single Cloudera Manager , single KDC instance

This is the simplest and easiest setup. All the Kafka clusters are managed by a single Cloudera Manager instance that uses a single KDC instance. In this case, the load balancer feature can be enabled in all the Kafka clusters and the rest is enabled automatically.

- Multiple Cloudera Manager, multiple KDC instances

In this case, all the Kafka clusters have their own separate Cloudera Manager and KDC instances. The load balancer feature can be enabled in all the Kafka clusters, but cross-realm trust should be set by the system administrator for the client principals to be able to authenticate to any of the Kafka services. The load balancer listeners log in into their own KDC using the load balancer principal, and the Kafka client needs a credential that can log in into any of the KDCs.

- Multiple Cloudera Manager, single KDC instances

If you use multiple Kafka clusters with multiple Cloudera Manager instances and yet they use the same single KDC, the feature does not work, because the CM servers invalidate each other's load balancer principal credentials in the keytab when getting the load balancer principal. Only one Cloudera Manager instance's Kafka cluster load balancer credentials will be fine, others will be stale. In this case, after enabling the feature as mentioned above, it is possible to override keytab or principal for the load balancer listener setting the following property as safety valve:

```
listener.name.lb.gssapi.sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required doNotPrompt=true useKeyTab=true storeKey=true keyTab="/tmp/loadbalancer.keytab" principal="load_balancer_principal";
```

The keytab can be created manually or the ones that are actually valid in one of the Kafka clusters can be copied to other cluster brokers. It is important to make sure that the process has permissions to read the keytab files because the permissions can change while copying keytabs from one host to another. The mentioned safety valve is also a way to override the load balancer principal if the generated one is not appropriate (or if the credentials are created manually and the administrator chose a different principal name than the default Kafka). Whichever option you use, it is important for the host part to be the same as load balancer host; otherwise, the forwarded requests will fail.

Setup for TLS/SSL encryption

If you are using TLS/SSL encryption, you need to select a method to resolve SSL hostname verification failure.

If TLS encryption is used and a client connects to the load balancer host, the SSL hostname verification fails on the Kafka client side, because the client compares the hostnames in the broker certificates with the actual hostnames that are used in bootstrap.servers for the connection.

You can use one of the following methods to prevent an SSL hostname verification failure.

- Using Subject Alternative Name (SAN) in the certificates

The optimal solution for the SSL hostname verification is to add the load balancer hostname as a SAN to the certificates of each broker.



Important: This is currently not done automatically with AutoTLS.

- Using wildcard certificates

If the load balancer and the brokers are in the same domain, you can also use wildcard certificates where it is not needed to enumerate the brokers and the load balancer one by one. Ensure you include the domain in the certificate.

- Disabling hostname verification on the client side

If modifying the certificates is a big effort, it is also possible to disable the hostname verification on the Kafka client side. The clients should include an empty string for the SSL algorithm:

```
ssl.endpoint.identification.algorithm=
```

Connecting to the Kafka cluster using load balancer

Learn how to connect a client to a Kafka cluster that is located behind a load balancer.

Before you begin

If you have set the Kafka Broker Load Balancer Listener Port property during broker configuration, ensure that you have also configured the load balancer to forward requests to the port number specified in Kafka Broker Load Balancer Listener Port.

Procedure

To connect to the Kafka cluster using the load balancer host and port, use the following command:

```
kafka-topics --list --bootstrap-server load_balancer_host:load_balancer_port
```

The load balancer is only used for connection bootstrap. The clients still need to be able to connect to the brokers directly. This also means that if the client has discovered a cluster and is actively using it and the cluster suddenly stops, the client tries to connect to the bootstrap servers it discovered in the bootstrap phase as metadata, so it does not automatically ask the load balancer for new bootstrap servers and it does not fail over to the other cluster even if the load balancer already routes new requests to the other cluster.

Configuring Kafka ZooKeeper chroot

By default, the /kafka path is used in ZooKeeper to store Kafka related metadata. This path can be changed by configuring the ZooKeeper Root Kafka property.

About this task

Complete the following steps to change the Kafka ZooKeeper chroot on an already existing service. You can also configure the ZooKeeper Root property when adding a new Kafka service to a cluster. The property can be configured on the Review Changes page when using the Add a Service wizard.



Important: Configuring the Kafka ZooKeeper chroot must be done during broker setup, before the broker is started for the first time. If the property is changed on an already running broker, metadata stored in the previously configured paths will not be available to Kafka once Kafka is restarted. This can lead to potential data loss.

Procedure

1. Select the Kafka service.
2. Go to Configuration and find the ZooKeeper Root property.
3. Add the path to use as a chroot environment for the Kafka cluster.
Cloudera recommends that you use /kafka.
4. Enter a Reason for change and click Save Changes.
5. Restart the Kafka service.

Results

The Kafka ZooKeeper chroot is configured. Kafka uses the configured path to store its metadata in ZooKeeper.

Kafka rack awareness

Learn about Kafka rack awareness and how it can be configured for Kafka brokers and clients.

Racks provide information about the physical location of a broker or a client. A Kafka deployment can be made rack aware by configuring rack awareness for the Kafka brokers and clients respectively. Enabling rack awareness can help in hardening your deployment, it provides durability guarantees for your Kafka service, and significantly decreases the chances of data loss.

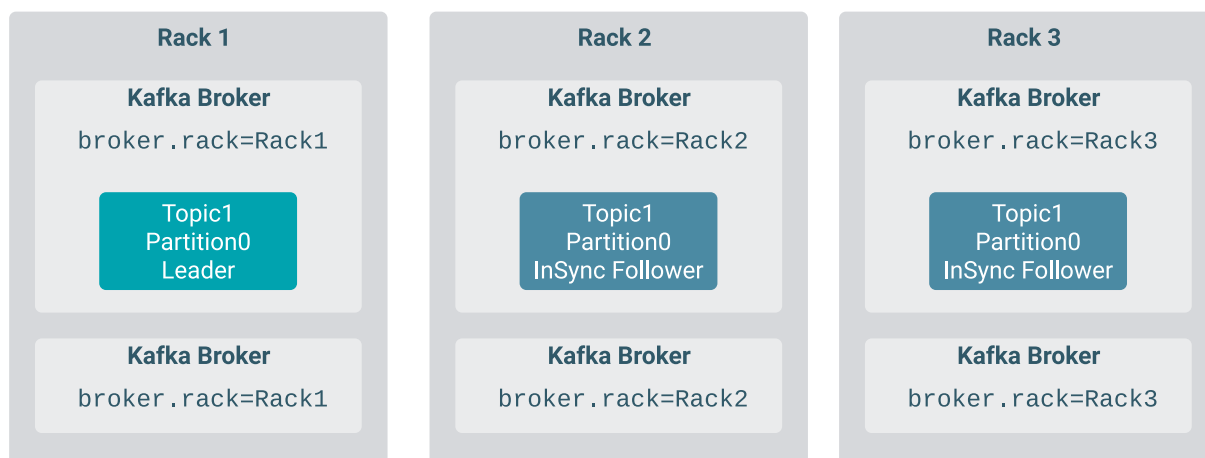
Rack awareness for Kafka brokers

Learn about Kafka broker rack awareness and how rack aware Kafka brokers behave.

To avoid a single point of failure, instead of putting all brokers into the same rack, it is considered a best practice to spread your Kafka brokers among racks. In cloud environments Kafka brokers located in different availability zones or data centers are usually deployed in different racks. Kafka brokers have built in support for this type of cluster topology and can be configured to be aware of the racks they are in.

If you create, modify, or redistribute a topic in a rack-aware Kafka deployment, rack awareness ensures that replicas of the same partition are spread across as many racks as possible. This limits the risk of data loss if a complete rack fails. Replica assignment will try to assign an equal number of leaders for each broker, therefore, it is advised to configure an equal number of brokers for each rack to avoid uneven load of racks.

For example, assume you have a topic partition with 3 replicas and have the brokers configured in 3 different racks. If rack awareness is enabled, Kafka will try to distribute the replicas among the racks evenly in a round-robin fashion. In the case of this example, this means that Kafka will ensure to spread all replicas among the 3 different racks, significantly decreasing the chances of data loss in case of a rack failure.



Configuring rack awareness for Kafka brokers

Learn how to configure rack awareness for Kafka brokers

About this task

Rack awareness is enabled and configured by specifying rack information for your hosts using the HostsAll HostsActions for SelectedAssign Rack action in Cloudera Manager and selecting the Enable Rack Awareness Kafka service property. Once selected, Enable Rack Awareness automatically configures racks for each Kafka broker based on the rack information you provided using the Assign Rack action.



Important: If after configuring and enabling rack awareness you make changes to rack information (for example, change a rack name), ensure that you restart the Kafka service. If the rack information is changed, the Kafka service will become stale, Cloudera Manager, however, will not display the Kafka service as stale.

Before you begin

- In order for rack awareness to properly function, the brokers in your deployment must be spread across available racks. If all brokers are deployed on the same rack, enabling and configuring rack awareness will not provide you with any benefits.

- If you previously configured and enabled rack awareness by manually configuring the `broker.rack` property with Kafka Broker Advanced Configuration Snippet (Safety Valve), ensure that you remove all `broker.rack` entries from the advanced configuration snippet. The advanced configuration snippet takes precedence over Enable Rack Awareness and overwrites the configuration set by Enable Rack Awareness.
- Specify rack information for your hosts. Complete [Specifying Racks for Hosts](#).

Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to Configuration.
3. Find and select the Enable Rack Awareness property.
4. Click Save Changes.
5. Restart the Kafka service.

Results

Rack awareness is enabled and configured for the Kafka brokers.

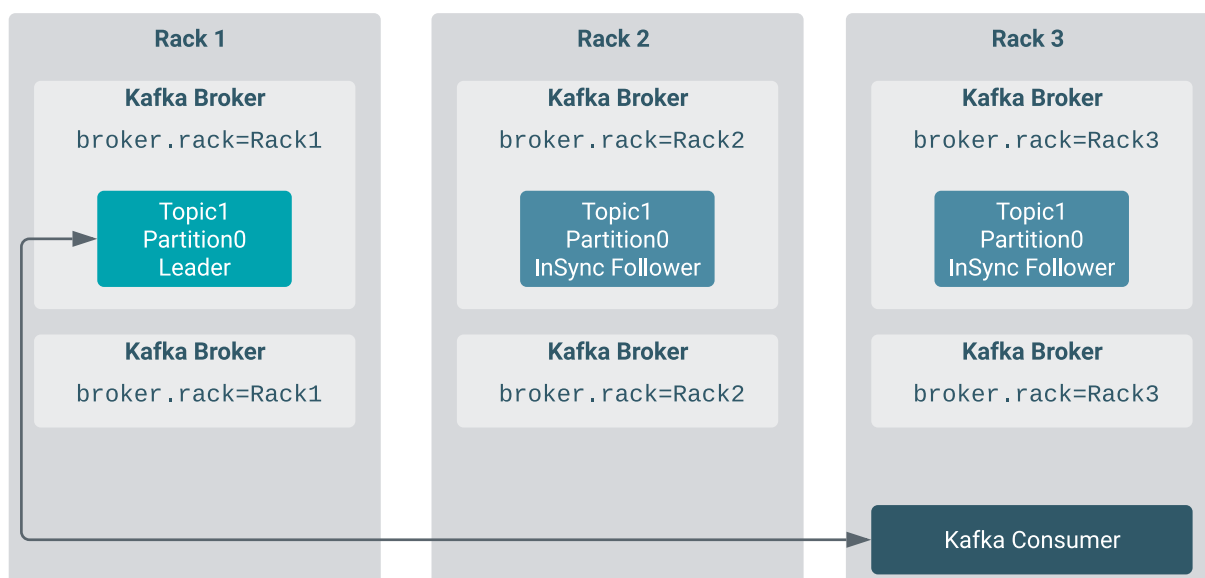
What to do next

Configure rack awareness for Kafka clients.

Rack awareness for Kafka consumers

Learn about leader fetching, which can be used to make Kafka consumers rack aware

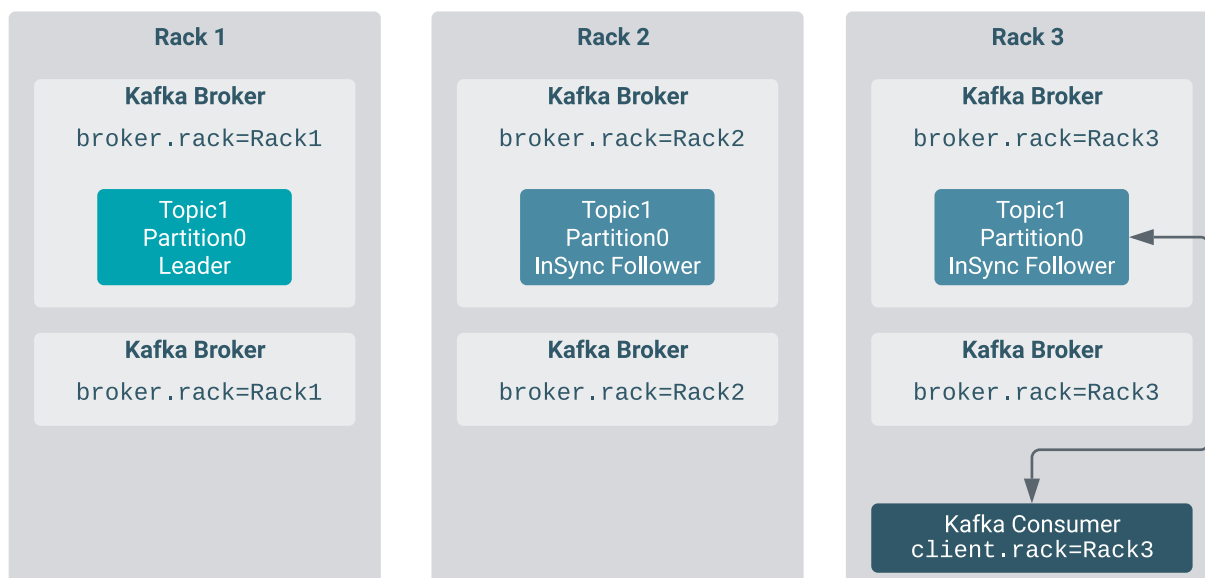
When a Kafka consumer tries to consume a topic partition, it fetches from the partition leader by default. If the partition leader and the consumer are not in the same rack, fetching generates significant cross-rack traffic, which has a number of disadvantages. For example, it can generate high costs and lead to lower consumer bandwidth and throughput.



For this reason, it is possible to provide the client with rack information so that the client fetches from the closest replica instead of the leader. If the configured closest replica does not exist (there is no replica for the needed partition in the configured closest rack), it uses the partition leader. This feature is called follower fetching and it can be used to mitigate the costs generated by cross-rack traffic or increase consumer throughput.



Note: Due to the nature of the Kafka protocol and high watermark propagation, consumers might experience increased message latency when fetching from a replica compared to when they are fetching from the leader.



Configuring rack awareness for Kafka consumers

Learn how to make Kafka consumers rack aware by enabling and configuring follower fetching.

About this task

Kafka Consumers can be made rack aware enabling follower fetching for your Kafka deployment. Follower fetching can be enabled by configuring `replica.selector.class` property for the broker and configuring the `client.rack` property in the consumer's configuration. The `replica.selector.class` property is not directly available for configuration in Cloudera Manager and you must use an advanced security snippet to configure it.

Before you begin

Ensure that brokers have rack awareness enabled. For more information, see [Configuring rack awareness for Kafka brokers](#).

Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to Configuration.
3. Find the Kafka Broker Advanced Configuration Snippet (Safety Valve) for `kafka.properties` property.
4. Add the following configuration entry to the advanced configuration snippet.

```
replica.selector.class=org.apache.kafka.common.replica.RackAwareReplicaSelector
```

5. Click Save Changes.
6. Restart the Kafka service.

7. Add the following to your consumer configuration.

```
client.rack=[***RACK ID***]
```

Replace `[***RACK ID***]` with the ID of the rack that the consumer is running in. The rack ID should match one of the rack ID's you configured for the brokers. Ensure that you configure each consumer and add its corresponding rack ID. If the consumer is deployed in a rack with no brokers, specify the rack ID of a broker that is closest to the rack that the consumer is running in.

Results

Follower fetching is enabled for the Kafka deployment. Kafka consumers are now rack aware and attempt to consume from the replica that is in the closet rack instead of consuming from the replica leader.

Rack awareness for Kafka producers

Learn about rack awareness for Kafka producers.

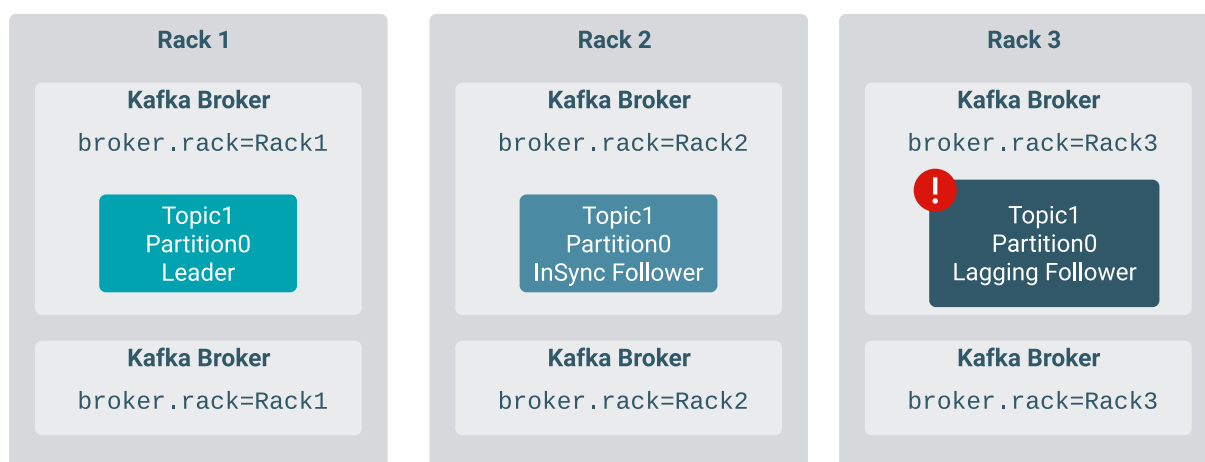
Compared to brokers or consumers, there are no producer specific rack-awareness features or toggles that you can enable. However, in a deployment where rack awareness is an important factor, you can make configuration changes so that producers make use of rack awareness and have messages replicated to multiple racks.

Specifically, Cloudera recommends a configuration that ensures that the produced messages are replicated to at least two different racks before the messages are considered to be successful. This involves configuring acks to all in the producer configuration and setting up `min.insync.replicas` for the topics in a way that ensures a minimum of two racks get the message before the produce request is considered successful.

The configuration of the acks property is fixed. If you want to make your producers rack aware, the property must be set to all no matter the cluster topology or deployment.

The exact value you set for `min.insync.replicas` on the other hand depends on your cluster deployment. Specifically, the `min.insync.replicas` value you must set will depend on the number of racks, brokers, and the replication factor of your topics. Cloudera recommends that you exercise caution and review the following examples to better understand configuration.

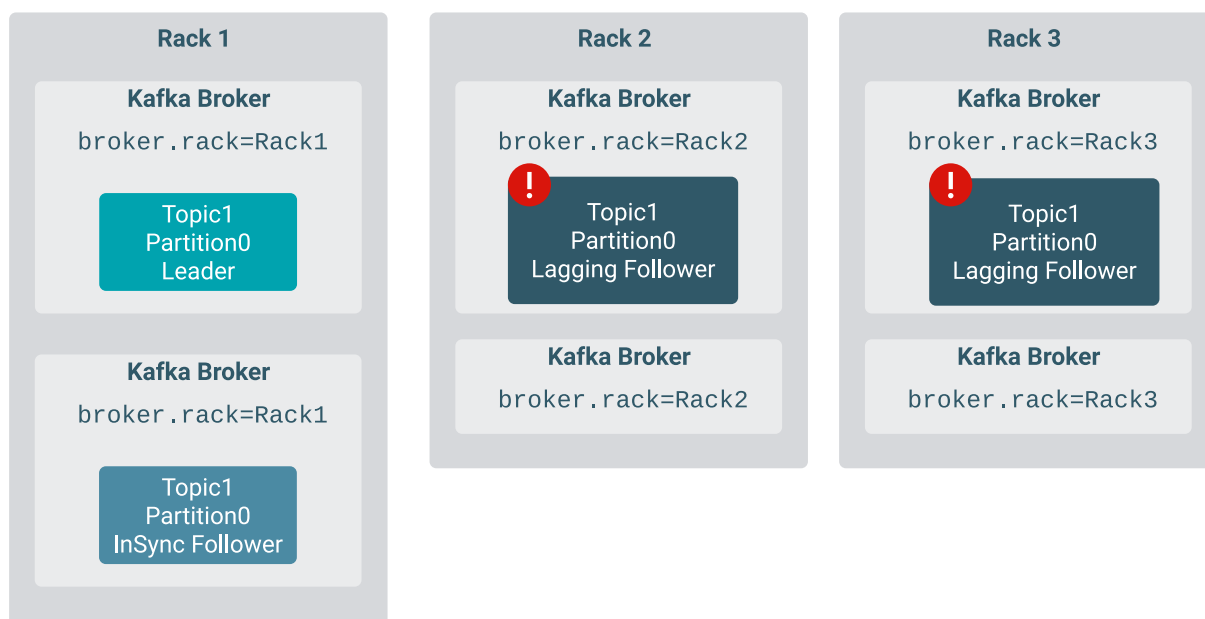
For example, consider a Cloudera recommended deployment that has three racks with topic replication set to 3. In a case like this, a `min.insync.replicas` setting of 2 ensures that you always have data written to at least two different racks even if one replica is lagging.



Understand however, that setting `min.insync.replicas` to 2 does not universally work for all deployments and may not guarantee that you always have your produced message in at least two racks. Configuration depends on the number of replicas, as well as the number of racks and brokers.

If you have more replicas and brokers than racks, you will have at least two replicas in the same rack. In a case like this, setting `min.insync.replicas` to 2 is not sufficient, a partition might become unavailable under certain circumstances.

For example, assume you have three racks with topic replication factor set to 4, meaning that there are a total of four replicas. Additionally, assume that only two of the replicas are in the in-sync replica set (ISR), the leader and one of the followers, and both are located in the same rack. The other two replicas are lagging. Unclean leader election is disabled to avoid data loss.



When the leader and the in-sync follower (located in the same rack) successfully append a produced message to the log, message production is considered successful. The leader does not wait for acknowledgement from the lagging replicas. This is because `acks=all` only guarantees that the leader waits for the replicas that are in the ISR (including itself). This means that while the latest messages are available on two brokers, both are located on the same rack. If the rack goes down at the same time or shortly after production is successful, the partition will become unavailable as only the two lagging replicas remain, which cannot become leaders.

In cases like this, a correct value for `min.insync.replicas` would be 3 instead of 2 as three ISRs would guarantee that messages are produced to at least two different racks.

Configuring rack awareness for Kafka producers

Learn how to enable and configure rack awareness for Kafka producers.

About this task

Enabling rack awareness for Kafka producers involves configuring your Kafka deployment in a way that ensures that producers commit messages to at least two separate brokers that are deployed on different racks. This can be done by configuring your producers to provide the highest available guarantee on message delivery and configuring `min.insync.replicas` for your topics.

Before you begin

Ensure that brokers have rack awareness enabled. For more information, see [Configuring rack awareness for Kafka brokers](#).

Procedure

1. Add the following to your producer configuration.

```
acks=all
```

This is the default configuration for producer version 3.0.0 or later. As a result, configuring this property might not be required.

2. Configure `min.insync.replicas` for the produced topics to a value that ensures the desired number of racks (minimum of 2) get the message before the produce request is considered successful.

Results

Rack awareness for Kafka producers is configured. Producers will now ensure that messages are produced to at least 2 (or more) of the available racks.