Cloudera Runtime 7.3.2

Configuring Apache Kafka

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

https://docs.cloudera.com/

Legal Notice

Contents

Installing Kafka in Cloudera Base on premises

Learn how to install Apache Kafka on an existing Cloudera Base on premises cluster. You can deploy Kafka in either KRaft or ZooKeeper mode.

Installing Kafka on an existing Cloudera Manager cluster involves adding the Kafka service to your cluster and configuring its roles. The Kafka service is added and configured using the **Add Service** wizard in Cloudera Manager.

KRaft versus ZooKeeper

You can deploy Kafka in either KRaft or ZooKeeper mode. However, Cloudera recommends that you deploy clusters in KRaft mode. This is because ZooKeeper-based clusters are deprecated. Additionally, support for ZooKeeper-based Kafka clusters will be removed in a future release.

KRaft offers enhanced reliability, scalability, and throughput over ZooKeeper. Metadata operations are more efficient as they are directly integrated.

Kafka Service Roles

The Kafka service in Cloudera Manager consists of multiple role types. During installation, you provision the roles that match your deployment requirements:

- **Kafka Broker**: The core data plane role responsible for handling client requests, storing data, and replicating partitions. A Kafka service requires at least one broker, though a minimum of three brokers is recommended for production deployments to ensure high availability and fault tolerance.
- **KRaft Controller**: The metadata management role used when deploying Kafka in KRaft mode. KRaft controllers form a Raft quorum that manages cluster metadata without requiring ZooKeeper. This role is only used when Kafka is configured to use KRaft as the metadata store.
- **Kafka Connect**: An optional role that provides a framework for connecting Kafka with external systems. Kafka Connect allows you to stream data between Kafka and other data systems using pre-built or custom connectors. For detailed information on provisioning, see Setting up Kafka Connect.
- **MirrorMaker**: An optional role used for mirroring data between Kafka clusters. MirrorMaker replicates topics from a source cluster to a destination cluster. For information on provisioning see Setting up MirrorMaker.

Important: MirrorMaker is deprecated and will be removed in a future release. Cloudera recommends you use Streams Replication Manager instead. See Streams Replication Manager Overview

Installing Kafka in KRaft mode

Learn how to add a Kafka service using KRaft to an existing Cloudera Manager cluster.

About this task

The following steps walk you through how you can add Kafka in KRaft mode to an already existing Cloudera Base on premises cluster. The steps are aimed to provide you with the basic process of installation and do not go into detail or provide recommendations on configuring service properties or security setup.

Tip: Although the following steps explicitly walk you through installation on an existing cluster using the **Add Service** wizard, you can also use the following steps as reference when installing a new cluster that will include Kafka.

Before you begin

- You must deploy a minimum of three Kafka Broker instances for production deployments to ensure high availability and fault tolerance.

- KRaft requires an odd number of controllers to function. You must always deploy an odd number of KRaft Controller service roles. Service setup fails if you try to deploy an even number of roles.
- KRaft can function with a single KRaft Controller role instance, but you must deploy a minimum of three for production use. Deploying a Kafka service with a single KRaft Controller is only recommended for development and testing purposes.
- Cloudera recommends that you deploy the KRaft Controller service roles on dedicated hosts. If deployment on dedicated hosts is not feasible, or if you are deploying a lightweight cluster where high availability is not a requirement, you can colocate the controllers on the same hosts as the brokers. In general, if your deployment can tolerate the simultaneous failure of two colocated nodes, then deploying the controllers and brokers on the same hosts is a viable option.
- The general hardware and deployment recommendations that exist for ZooKeeper also hold true for KRaft Controllers. For more information, see Performance considerations.

Procedure

1. In Cloudera Manager, select the cluster where you want to install Kafka.
2. Click Actions Add Service .
3. Select Kafka from the list of services and click Continue.
4. Select service dependencies.

   Important: Selecting a dependency option that includes ZooKeeper does not configure Kafka to use ZooKeeper as its metadata service. Metadata service selection happens in a later step.

   The **Select Dependencies** page presents dependencies as radio button options. The available dependency options differ based on what services are installed on your cluster and the dependencies between them.

   For a basic KRaft Kafka deployment without authorization, select No Optional Dependencies. If you need additional functionality such as authorization or audit logging, review the following guidelines to understand which dependency option to select:

   - **Ranger (optional)**: Select a dependency option that includes Ranger if you need fine-grained authorization for Kafka. If you select Ranger, you can configure authorization policies through the Ranger UI. Ranger requires Solr for short-term audit log storage (up to 90 days).
   - **HDFS and CORE_SETTINGS (optional)**: Select a dependency option that includes HDFS and CORE_SETTINGS only if you have selected Ranger and want to store Ranger audit logs in HDFS for long-term retention (beyond 90 days). HDFS and CORE_SETTINGS are always bundled together because HDFS requires the core Hadoop configuration to function. Kafka itself does not require HDFS.
   - **ZooKeeper (optional)**: When deploying Kafka in KRaft mode, Kafka does not use ZooKeeper for metadata management. However, you might need to select a dependency option that includes ZooKeeper if other services you are integrating with (such as HDFS or Ranger) require ZooKeeper for their own operation. In this case, ZooKeeper is used by those other services, not by Kafka.
5. Click Continue.
6. Assign role instances to hosts.

   On the **Assign Roles** page, you select which hosts will run each Kafka role. For a KRaft-based deployment, you must assign **Kafka Broker** and **KRaft Controller** roles. You can optionally assign **Kafka Connect** and **MirrorMaker** roles as well.

   a) Click the field below the role name to display a dialog containing a list of hosts.
   b) Select one or more hosts for the role and click OK.

   For **KRaft Controller** roles, you must select an odd number of hosts (minimum three recommended). For **Kafka Broker** roles, select at least three hosts for production deployments.
7. Click Continue.
8. On the **Review Changes** page, configure Kafka service properties.
   a) Ensure that the Kafka Metadata Store Service property is set to KRaft.

This property determines whether Kafka uses KRaft or ZooKeeper for metadata management. The default value is KRaft.

b) Review and configure other service properties as needed for your cluster and requirements.

9. Click Finish.

Results

The Kafka service is added to your cluster in KRaft mode. The Kafka service uses KRaft for metadata management

Related Information

Setting up Kafka Connect
Setting up MirrorMaker
Securing Apache Kafka
Kafka Properties in Cloudera Runtime

Installing Kafka in ZooKeeper mode

Learn how to add a Kafka service that uses ZooKeeper for metadata management.

Important: ZooKeeper-based Kafka deployments are deprecated. Cloudera recommends deploying new clusters in KRaft mode. Support for ZooKeeper-based Kafka deployments will be removed in a future Cloudera Base on premises release.

About this task

The following steps walk you through how you can add Kafka in ZooKeeper mode to an already existing Cloudera Base on premises cluster. The steps are aimed to provide you with the basic process of installation and do not go into detail or provide recommendations on configuring service properties or security setup.

Tip: Although the following steps explicitly walk you through installation on an existing cluster using the **Add Service** wizard, you can also use the following steps as reference when installing a new cluster that will include Kafka.

Before you begin

- You must have a ZooKeeper service installed and running on your cluster. ZooKeeper is a required dependency for ZooKeeper-based Kafka deployments.
- You must deploy a minimum of three Kafka Broker instances for production deployments to ensure high availability and fault tolerance.
- ZooKeeper-based Kafka cluster can be migrated to KRaft. For more information, see Migrating Kafka from ZooKeeper to KRaft.

Procedure

1. In Cloudera Manager, select the cluster where you want to install Kafka.
2. Click Actions Add Service .
3. Select Kafka from the list of services and click Continue.
4. Select service dependencies.

Important: Selecting a dependency option that includes ZooKeeper does not configure Kafka to use ZooKeeper as its metadata service. Metadata service selection happens in a later step where you must explicitly set the Kafka Metadata Store Service property to Zookeeper.

The **Select Dependencies** page presents dependencies as radio button options. The available options differ based on what services are installed on your cluster and the dependencies between them.

For a basic ZooKeeper-based Kafka deployment without authorization, select the option that includes only ZooKeeper. If you need additional functionality such as authorization or audit logging, review the following guidelines to understand which dependency option to select:

- **ZooKeeper (required)**: For ZooKeeper-based Kafka, you must select a dependency option that includes ZooKeeper, as it is required for Kafka's metadata management.
- **Ranger (optional)**: Select a dependency option that includes Ranger if you need fine-grained authorization for Kafka. If you select Ranger, you can configure authorization policies through the Ranger UI. Ranger requires Solr for short-term audit log storage (up to 90 days) and ZooKeeper for its own operation.
- **HDFS and CORE_SETTINGS (optional)**: Select a dependency option that includes HDFS and CORE_SETTINGS only if you have selected Ranger and want to store Ranger audit logs in HDFS for long-term retention (beyond 90 days). HDFS and CORE_SETTINGS are always bundled together because HDFS requires the core Hadoop configuration to function. Additionally, HDFS requires ZooKeeper for high availability features. Kafka itself does not require HDFS.

5. Click Continue.
6. Assign role instances to hosts.

   On the **Assign Roles** page, you select which hosts will run each Kafka role. For a ZooKeeper-based deployment, you must assign **Kafka Broker** roles. You can optionally assign **Kafka Connect** and **MirrorMaker** roles as well.

   a) Click the field below the role name to display a dialog containing a list of hosts.
   b) Select one or more hosts for the role and click OK.

   For **Kafka Broker** roles, select at least three hosts for production deployments. Do not assign **KRaft Controller** roles when deploying in ZooKeeper mode.
7. Click Continue.
8. On the **Review Changes** page, configure Kafka service properties.
   a) Find and set the Kafka Metadata Store Service property to Zookeeper.

      This property determines whether Kafka uses KRaft or ZooKeeper for metadata management. The default value is **KRaft**. You must change it to **Zookeeper**.
   b) Review and configure other service properties as needed for your cluster and requirements.
9. Click Finish.

Results

The Kafka service is added to your cluster in ZooKeeper mode. Kafka uses ZooKeeper for metadata management.

Related Information

Setting up Kafka Connect
Setting up MirrorMaker
Securing Apache Kafka
Kafka Properties in Cloudera Runtime

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 KafkaConfigurationMaximum Process File Descriptors and setting the required value. Cloudera recommends a configuration of 100000 or higher.

Important:

After changing the maximum process file descriptor limit, you might find that service roles are still limited by the number of file descriptors. Raising the maximum process file descriptors above the Linux kernel file descriptor limit will have no effect. Check the Linux kernel file descriptor limit on every host in the cluster and raise that if necessary.

You can find the Linux kernel file descriptor limit by running the following command on the Linux command line:

```
sudo cat /proc/sys/fs/nr_open
```

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 cluster for the Kafka metadata service. Depending on your deployment mode, this means either a dedicated ZooKeeper cluster (for ZooKeeper-based deployments) or dedicated KRaft controllers (for KRaft mode). 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 or containers. Kafka is sufficiently sensitive to I/O throughput and network performance that virtualized environments can impact broker operation. When running Kafka in a virtual environment, ensure adequate I/O performance and resource allocation. You may need to work with your infrastructure team or cloud provider to optimize Kafka performance in virtualized deployments.

Do not run other processes with brokers, ZooKeeper, or KRaft controllers

Due to I/O contention with other processes, it is generally recommended to avoid running other processes on the same hosts as Kafka brokers. Similarly, for optimal performance, ZooKeeper nodes and KRaft controllers should not be colocated with brokers or other services.

Keep the metadata service connection stable

Kafka relies heavily on having a stable connection to its metadata service. Putting an unreliable network between Kafka brokers and the metadata service (ZooKeeper or KRaft controllers) will cause connectivity issues and cluster instability. Examples of configurations to avoid:

- Do not put Kafka brokers and metadata service nodes on separated networks

- Do not put Kafka brokers and metadata service nodes on the same network with other high network loads

KRaft controller deployment considerations

When running Kafka in KRaft mode, follow these additional recommendations:

- For redundancy, a Kafka cluster should use 3 or more controllers. The number of controllers depends on factors like cost and the number of concurrent failures your system should withstand without availability impact. To withstand N concurrent failures, the KRaft controller cluster must include 2N + 1 controllers.
- KRaft controllers store all cluster metadata in memory and on disk. For a typical Kafka cluster, 5GB of main memory and 5GB of disk space for the metadata log directory is sufficient.

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 ((num.io.threads + 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 form 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_percentag
e=200' --entity-type users --entity-name [***USER NAME***] --entity-type c
lients --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_percentag
e=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_percentag
e=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_percentag
e=200' --entity-type users --entity-name [***USER NAME***] --entity-type c
lients --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_percentag
e=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_percentag
e=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  --e
ntity-type users --entity-name [***USER NAME***] --entity-type clients --ent
ity-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 hte 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 --bootstrap-server HOSTNAME:PORT --topics-to-mov
   e-json-file TOPICS TO MOVE.json --broker-list BROKER 1, BROKER 2 --gener
   ate
   ```

   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 --bootstrap-server HOSTNAME:PORT  --reas
   signment-json-file REASSIGNMENT CONFIGURATION.json --bootstrap-se
   rver 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 --bootstrap-server HOSTNAME:PORT --reass
ignment-json-file REASSIGNMENT CONFIGURATION.json  --bootstrap-serve
r 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 checks during a rolling restart to ensure that Kafka roles stay healthy. Rolling restart checks are available for both Kafka brokers and KRaft controllers, with each configured independently to provide different levels of health guarantees.

Rolling restart checks are enabled by default to ensure cluster health during rolling restarts. For Kafka brokers, the default check ensures that healthy partitions stay healthy during restarts. For KRaft controllers, the default check ensures that the majority of controllers remain online during restarts. These checks help prevent service disruptions that could occur if roles are restarted without verifying cluster health. For example, without checks, Cloudera Manager might restart a broker while the previous broker is not fully ready for operation, causing outages and corrupted log indexes. Similarly, for KRaft controllers, restarting without proper checks can disrupt the metadata quorum and impact cluster operations.

Broker and controller rolling restart checks are configured independently using separate properties. This allows you to disable, enable, or configure different check levels for each type based on your cluster requirements.

Rolling restart checks are available for both Kafka brokers and KRaft controllers:

- **Kafka broker checks** are configured using the Cluster Health Guarantee During Rolling Restart property. These checks focus on partition health and replica synchronization to ensure that topics remain accessible and data is not lost during broker restarts.
- **KRaft controller checks** are configured using the KRaft Cluster Health Guarantee During Rolling Restart property. These checks focus on maintaining metadata quorum health to ensure that the majority of controllers remain available during controller restarts. These checks are only applicable when KRaft is used as the metadata store.

Rolling restart checks for Kafka brokers

Kafka broker rolling restart checks focus on partition health and replica synchronization. There are multiple checks available, each providing a different 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 five 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

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 (default)

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.

all partitions fully replicated

This setting ensures that all partitions are in a fully synchronized 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 out of sync. Additionally, this setting ensures that the restarted broker is accepting requests on its service port before the next broker is restarted.

When Cloudera Manager executes a broker 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.

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*.

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.

Rolling restart checks for KRaft controllers

KRaft controller rolling restart checks are configured independently from Kafka broker rolling restart checks using the KRaft Cluster Health Guarantee During Rolling Restart property. KRaft controller rolling restart checks only apply when KRaft is used as the metadata store for the Kafka service.

The KRaft controller rolling restart checks ensure that the KRaft quorum remains healthy during a rolling restart. The checks verify that the majority of controllers (half + 1) remain available and operational throughout the restart process. Unlike broker checks which focus on partition health, controller checks focus on maintaining metadata quorum health.

The KRaft Cluster Health Guarantee During Rolling Restart property has two settings:

none

This setting disables KRaft controller rolling restart checks. If this option is selected, no checks are performed on controllers during a rolling restart.

majority of controllers online (default)

This setting ensures that during a rolling restart, the majority of KRaft controllers remain online and operational. This is the recommended setting to maintain metadata availability during restarts.

KRaft controller rolling restart checks use the kafka-metadata-quorum command line tool to verify quorum health. The checks are performed in two phases:

- **Pre-check (before stopping a controller)**: Before stopping a controller, Cloudera Manager verifies that the majority of the KRaft quorum will remain available after the controller is stopped. The check counts the number of active controllers (controllers with low lag and recent metadata fetch timestamps) and ensures that at least half + 1 controllers will remain active after stopping the current controller. If the majority requirement cannot be met, the check waits and retries until the requirement is satisfied or a timeout is reached.
- **Post-check (after starting a controller)**: After restarting a controller, Cloudera Manager verifies that the controller is actively participating in metadata updates. The check monitors the controller's LastFetchTimestamp and ensures it is increasing, which indicates the controller is successfully fetching and processing metadata updates from the quorum leader. The check succeeds when the timestamp increases.

A controller is considered active if it meets the following criteria:

- Its lag (number of uncommitted metadata messages) is below the configured threshold (default: 5 messages)
- Its LastFetchTimestamp is recent (within the configured threshold, default: 5 seconds)
- Its status is either Leader or Follower (not Observer)

In addition to the main KRaft Cluster Health Guarantee During Rolling Restart property, several configuration properties are available to fine-tune the behaviour of KRaft controller rolling restart checks:

- Maximum Allowed Runtime for Kafka Controller Rolling Restart Checks - Specifies the overall timeout (default: 15 minutes)
- Retry Interval for Kafka Controller Rolling Restart Checks - Specifies the wait time between check attempts (default: 30 seconds)
- API Timeout For kafka-metadata-quorum Client Used In Kafka Controller Rolling Restart Checks - Specifies the timeout for the kafka-metadata-quorum command (default: 60 seconds)
- Maximum Allowed Lag for Active Kafka Controllers - Specifies the maximum lag in messages before a controller is considered inactive (default: 5 messages)
- Maximum Allowed Lag in LastFetchTimestamp for Active Kafka Controllers - Specifies the maximum timestamp lag before a controller is considered inactive (default: 5 seconds)

Note: KRaft controller rolling restart checks do not require manual client configuration. The checks use the kafka-metadata-quorum tool which connects directly to the KRaft controllers using the cluster's internal configuration.

Advanced configuration for broker checks

There are two scenarios when additional configuration is required for Kafka broker rolling restart checks. 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 for brokers

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.

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.

Note: If Kafka brokers are configured with JVM properties that specify a unique port, such as when configuring a JMX agent, ensure that you configure the Kafka Broker JVM properties in the Additional Broker Java Options configuration item instead of adding the properties using environment variables. This prevents rolling restart check failures caused by port collisions.

Configuring rolling restart checks for controllers

You can configure Cloudera Manager to enable or disable checks on KRaft controllers during a rolling restart using the KRaft Cluster Health Guarantee During Rolling Restart property. Additionally, you can fine-tune the behavior of these checks using various timeout and threshold properties.

Before you begin

KRaft controller rolling restart checks are only applicable when KRaft is used as the metadata store for the Kafka service.

Procedure

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

   Select none to disable checks. Otherwise, to have checks enabled, leave it at the default majority of controllers online to maintain metadata quorum availability during rolling restarts.
4. Fine-tune KRaft controller rolling restart check behaviour by configuring the following properties:

   - Maximum Allowed Runtime for Kafka Controller Rolling Restart Checks
   - Retry Interval for Kafka Controller Rolling Restart Checks
   - API Timeout For kafka-metadata-quorum Client Used In Kafka Controller Rolling Restart Checks
   - Maximum Allowed Lag for Active Kafka Controllers
   - Maximum Allowed Lag in LastFetchTimestamp for Active Kafka Controllers

   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

KRaft controller rolling restart checks are configured. The configuration is used during subsequent rolling restarts.

Configuring the client configuration used for broker rolling restart checks

Cloudera Manager requires Kafka client configuration files to perform rolling restart checks on brokers. 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 requir
ed useKeyTab=true storeKey=true keyTab="[***PATH TO KEYTAB***]" princi
pal="[***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 for brokers*.

Related Information

Streams Messaging

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.

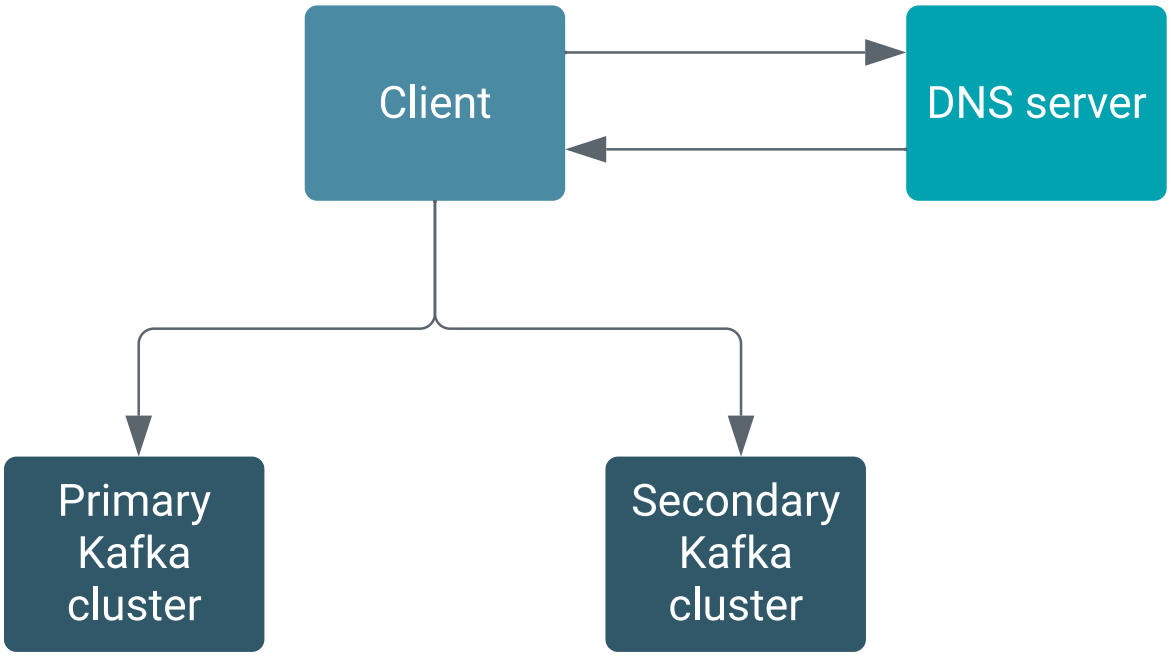
A common architectural pattern for Kafka is to have redundant Kafka clusters across multiple data centers so that applications can either load balance or fail over to the remote data center in case the local one becomes unavailable. Finding a suitable cluster by the client is called cluster discovery. 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.
```

It 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.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.

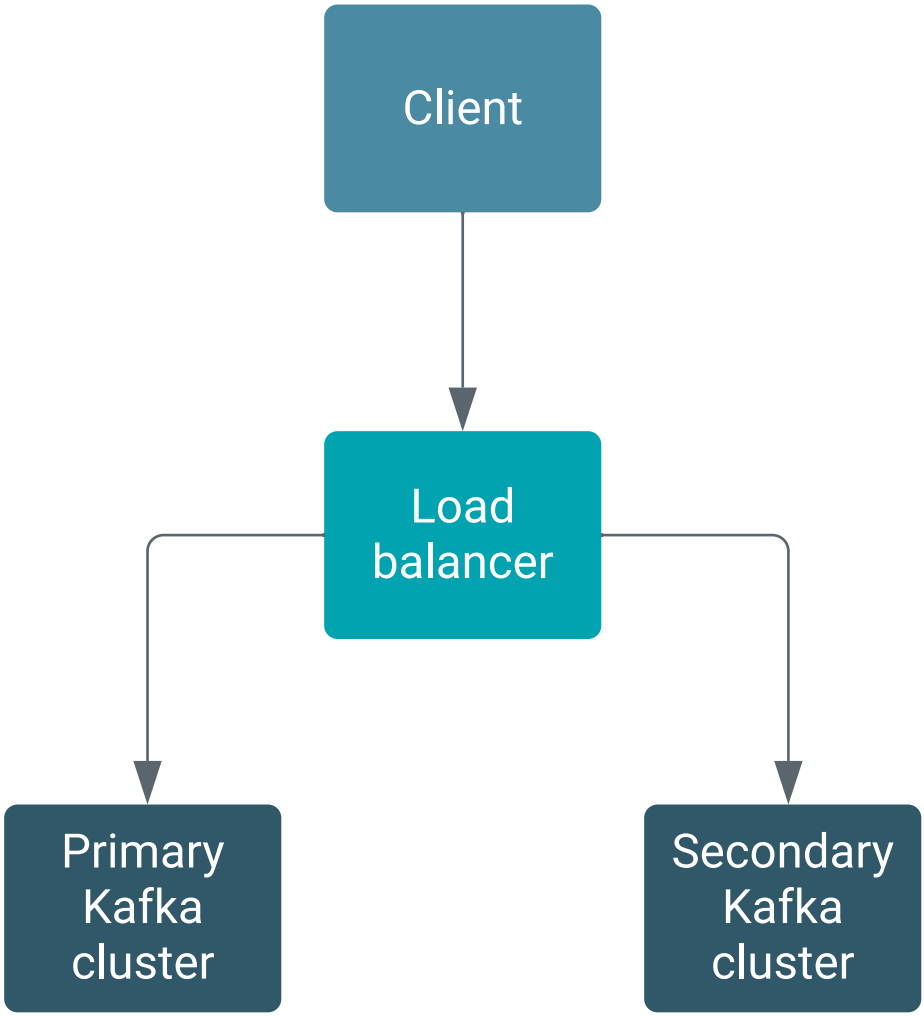
In case of a broker failure, you can change the records in the DNS server to point to other brokers. The next time a client tries to connect, it resolves to a different cluster's broker. This way you do not hardcode broker FQDNs to the bootstrap server list.

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.Kr
b5LoginModule
        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.Kr
  b5LoginModule required doNotPrompt=true useKeyTab=true storeKey=true key
  Tab="/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.

Configure Kafka rack awareness

Learn how to configure rack awareness and multi-level rack awareness 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.

The following tasks give step by step instructions on how you can configure and enable rack awareness and multi-level rack awareness for Kafka brokers and clients. The following covers configuration only. If you want to learn more about the rack awareness feature and how rack-aware Kafka deployments behave, see Kafka rack awareness.

Note: Although the feature is called rack awareness, the term rack does not necessarily mean an actual physical server rack. Instead, a rack from Kafka's perspective represents a physical location or independent physical infrastructure. For example, in many production deployments, the feature is used to specify the individual Data Centers that the brokers and clients are running in.

Configuring rack awareness for Kafka brokers

Learn how to configure rack awareness for Kafka brokers.

About this task

Rack awareness can be enabled and configured by specifying rack IDs. This can be done in two ways.

You can set the rack IDs on the level of the host machine in Cloudera Manager and configure Kafka to use the rack IDs specified for the host machines. Rack IDs are specified with the Hosts All Hosts Actions for Selected Assign Rack action in Cloudera Manager. Kafka can be configured to use these IDs by selecting the Enable Rack Awareness property.

Alternatively, you can manually specify the rack ID for each broker role using the Broker Rack ID property. In this case, selecting Enable Rack Awareness is not required.

If IDs are specified in Broker Rack ID and Enable Rack Awareness is selected, the ID's specified in Broker Rack ID take precedence.

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 and Cruise Control services. If the rack information is changed, the services become stale, Cloudera Manager, however, does not display the services 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 and overwrites the configuration set by both Enable Rack Awareness and Broker Rack ID.
- If you want to specify the rack IDs on the level of the host, you must specify rack information for your hosts before you start the following procedure. Complete Specifying Racks for Hosts.
- If you are using Broker Rack ID to set rack IDs, ensure that you set the property for each broker instance separately. Do not specify this property on a Kafka service level.

Procedure

1. In Cloudera Manager, select the Kafka service.
2. Configure rack IDs.

   For Use the IDs set on the host level

   a. Go to Configuration.
   b. Find and select the Enable Rack Awareness property.
   c. Click Save Changes.

   For Set IDs on a broker level

Repeat the following steps for each Kafka broker role instance.

a. Go to Instances.
b. Click any of the broker role instances and go to Configuration.
c. Find and configure the Broker Rack ID property.

Add the ID of the rack (or other physical infrastructure) that the broker is deployed in. For example, rack1 or DC1

d. Click Save Changes.

3. 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.

Configuring multi-level rack awareness for brokers

Learn how to enable and configure multi-level rack awareness for brokers.

About this task

Multi-level rack awareness can be enabled and configured by specifying multi-level rack IDs and selecting the Enable multi-level rack awareness Kafka service property.

A multi-level rack ID has a different format than a standard rack ID. It is an absolute path, it starts with a slash (/), but does not end in a slash. For example, /uswest/R0 or /DC1/Rack1/HostA. The ID can be specified in two ways.

You can set the rack IDs on the level of the host machine in Cloudera Manager and configure Kafka to use the rack IDs specified for the host machines. Rack IDs are specified with the HostsAll HostsActions for SelectedAssign Rack action in Cloudera Manager. Kafka can be configured to use these IDs by selecting the Enable Rack Awareness property.

Alternatively, you can manually specify the rack ID for each broker role using the Broker Rack ID property. In this case, selecting Enable Rack Awareness is not required.

If IDs are specified in Broker Rack ID and Enable Rack Awareness is selected, the ID's specified in Broker Rack ID take precedence.

AlI IDs you specify for your brokers must have an identical number of levels in the ID.

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 and Cruise Control services. If the rack information is changed, the services become stale, Cloudera Manager, however, does not display the services as stale.

Before you begin

- In order for multi-level rack awareness to properly function, the topology of your deployment must be compatible. That is, the brokers in your deployment must be spread across available physical infrastructure (for example DCs and racks). If your deployment topology is not compatible, enabling and configuring multi-level 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 and overwrites the configuration set by both Enable Rack Awareness and Broker Rack ID.

- If you want to specify the rack IDs on the level of the host, you must specify rack information for your hosts before you start the following procedure. Complete Specifying Racks for Hosts.
- If you are using Broker Rack ID to set rack IDs, ensure that you set the property for each broker instance separately. Do not specify this property on a Kafka service level.

Procedure

1. In Cloudera Manager, select the Kafka service.
2. Configure rack IDs.

   For Use the IDs set on the host level

   a. Go to Configuration.
   b. Find and select the Enable Rack Awareness property.
   c. Click Save Changes.

   For Set IDs on a broker level

   Repeat the following steps for each Kafka broker role instance.

   a. Go to Instances.
   b. Click any of the broker role instances and go to Configuration.
   c. Find and configure the Broker Rack ID property.

      Add the ID of the rack (or other physical infrastructure) that the broker is deployed in. Ensure that you set multi-level IDs. For example, /uswest/R0 or /DC1/Rack1/HostA.
   d. Click Save Changes.
3. Find and select the Enable multi-level rack awareness property.
4. Restart the Kafka service.

Results

Multi-level rack awareness is enabled for the broker. The brokers create replica assignments with multi-level rack awareness guarantees.

What to do next

Configure rack awareness for Kafka clients.

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 the 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.

Configuring multi-level rack awareness for consumers

Learn how to configure consumer follower fetching in Kafka deployment that has multi-level rack awareness enabled.

About this task

If multi-level rack awareness is enabled for the brokers, you can specify a multi-level rack ID for the consumers in their configuration. If a multi-level rack ID is specified for the consumer, the consumer will fetch messages from the closest replica in the multi-level hierarchy based on the ID it is provided. The rack ID can be set with the client.rack property in the consumer's configuration.

Note: Unlike standard rack awareness and follower fetching, manually configuring a replica selector class in the broker's configuration is not required. If multi-level rack awareness is enabled for the broker, the required ReplicaSelector class is automatically installed.

Before you begin

Ensure that brokers have multi-level rack awareness enabled. For more information, see Configuring multi-level rack awareness for brokers.

Procedure

1. Choose an appropriate multi-level rack ID for the consumer.

   A multi-level rack ID is an absolute path, it starts with a slash (/) but does not end in a slash. The exact ID that you specify depends on the levels of hierarchy in your deployment (cluster topology) and the rack IDs specified for the broker.

   With standard rack awareness and follower fetching, the IDs you specify for the consumer must exactly match the ID specified for the brokers. With multi-level rack awareness, this is not the case. The levels of hierarchy represented in the ID of the consumer can differ compared to the broker rack IDs. Only a partial match is required to find a closest replica.

   For example, assume you have a two level hierarchy, DCs on the top level, racks on the second level. Additionally, assume there are two replicas. They are located in /DC1/R1 and /DC2/R3. Depending on the configuration of client.rack, the consumer will select different replicas. The behavior is as follows:

- If client.rack is /DC1, the /DC1/R1 replica is selected.
- If client.rack is /DC2/R5, the /DC2/R3 replica is selected.
- If client.rack is /DC2/R3, the /DC2/R3 replica is selected.
- If client.rack is not set, the leader is selected.
- If client.rack is random_id, the leader is selected.
- If client.rack is DC3, the leader is selected.

2. Set the client.rack property.

   For example:

   ```
   client.rack=/DC1/R1
   ```

Results

Consumer IDs are configured. The consumers will fetch data from the closest replica in the multi-level deployment.

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.insy nc.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.