Cloudera Runtime 7.1.7

Configuring Apache Kafka

Date published: 2019-12-18
Date modified: 2021-08-05

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

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

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

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

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

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

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

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

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

Contents

Operating system requirements 4

Performance considerations 4

Quotas 5

JBOD 5
- JBOD setup 6
- JBOD Disk migration 7

Setting user limits for Kafka 9

Configuring Kafka ZooKeeper chroot 9

Kafka rack awareness 10
- Rack awareness for Kafka brokers 10
- Configuring rack awareness for Kafka brokers 11
- Rack awareness for Kafka consumers 12
- Configuring rack awareness for Kafka consumers 13
- Rack awareness for Kafka producers 14
- Configuring rack awareness for Kafka producers 15

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.

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 Quotas and how to set them.

For a quick video introduction to quotas, see 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, multi-tenant clusters where a small set of clients using high volumes of data can degrade the user experience.

Quotas are byte-rate thresholds, defined per client ID. A client ID logically identifies an application making a request. A single client ID can span multiple producer and consumer instances. The quota is applied for all instances as a single entity. For example, if a client ID has a produce quota of 10 MB/s, that quota is shared across all instances with that same ID.

When running Kafka as a service, quotas can enforce API limits. By default, each unique client ID receives a fixed quota in bytes per second, as configured by the cluster (quota.producer.default, quota.consumer.default). This quota is defined on a per-broker basis. Each client can publish or fetch a maximum of *X* bytes per second per broker before it gets throttled.

The broker does not return an error when a client exceeds its quota, but instead attempts to slow the client down. The broker computes the amount of delay needed to bring a client under its quota and delays the response for that amount of time. This approach keeps the quota violation transparent to clients (outside of client-side metrics). This also prevents clients from having to implement special backoff and retry behavior.

You can override the default quota for client IDs that need a higher or lower quota. The mechanism is similar to per-topic log configuration overrides. Write your client ID overrides to ZooKeeper under /config/clients. All brokers read the overrides, which are effective immediately. You can change quotas without having to do a rolling restart of the entire cluster.

By default, each client ID receives an unlimited quota. The following configuration sets the default quota per producer and consumer client ID to 10 MB/s.

```
quota.producer.default=10485760
quota.consumer.default=10485760
```

To set quotas using Cloudera Manager, open the Kafka Configuration page and search for Quota. Use the fields provided to set the Default Consumer Quota or Default Producer Quota.

Related Information

Changing the Configuration of a Service or Role Instance

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 --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 --e
   xecute
   ```

   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 --v
erify
```

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.

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.

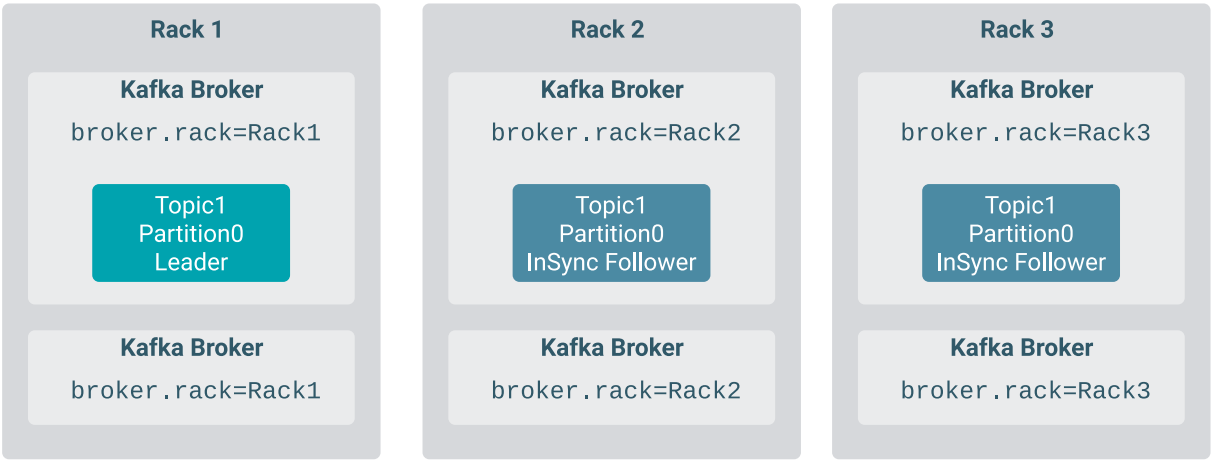
Rack awareness for Kafka brokers

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 for brokers by using the broker.rack property. This property is not directly available for configuration in Cloudera Manager and you must use an advanced security snippet to configure it. The value you set for the broker.rack property can be any user specified string.

Important: Do not use the HostsAll HostsActions for selectedAssign Rack action in Cloudera Manager to specify Kafka broker rack information. The rack information set using the action is not applied for the Kafka service.

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.
- Rack information must be configured separately for each broker. Do not set broker.rack globally for all brokers.

Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to Instances.
3. Configurebroker.rack using an advanced configuration snippet.

   Repeat the following steps for each Kafka broker role.

   a) Click on a Kafka broker role.
   b) Go to Configuration.
   c) Find the Kafka Broker Advanced Configuration Snippet (Safety Valve) for kafka.properties property and add the broker.rack property.

      For example:

      ```
      broker.rack=Rack1
      ```

   d) Click Save Changes.
4. Restart the Kafka service.

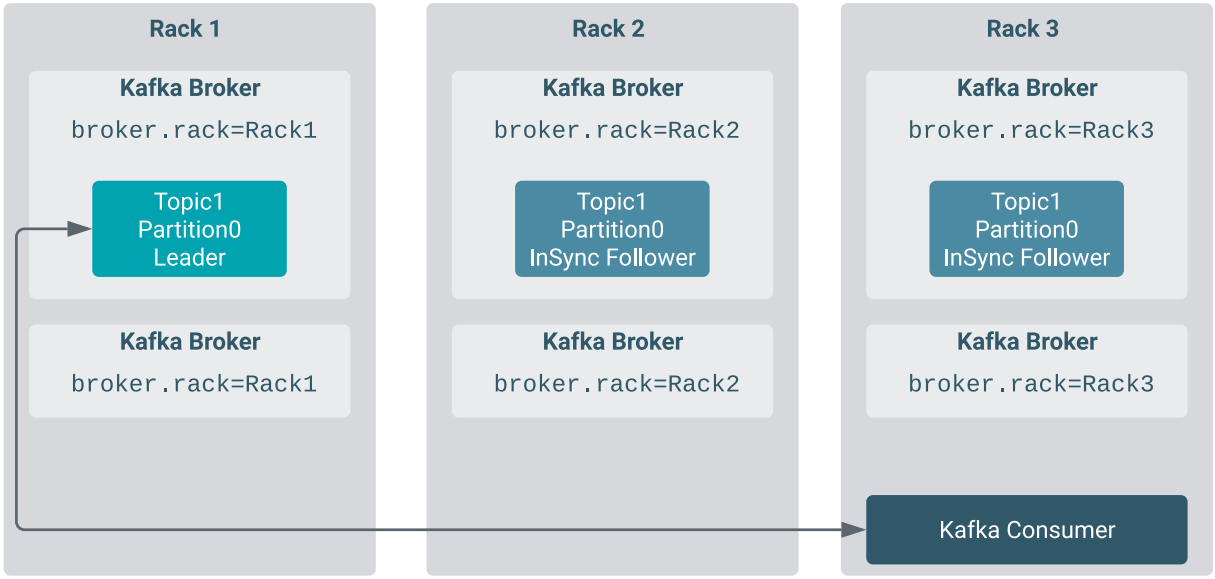
Results
Rack awareness is enabled and configured for the Kafka brokers.

What to do next
Configure rack awareness for Kafka clients.

Rack awareness for Kafka consumers

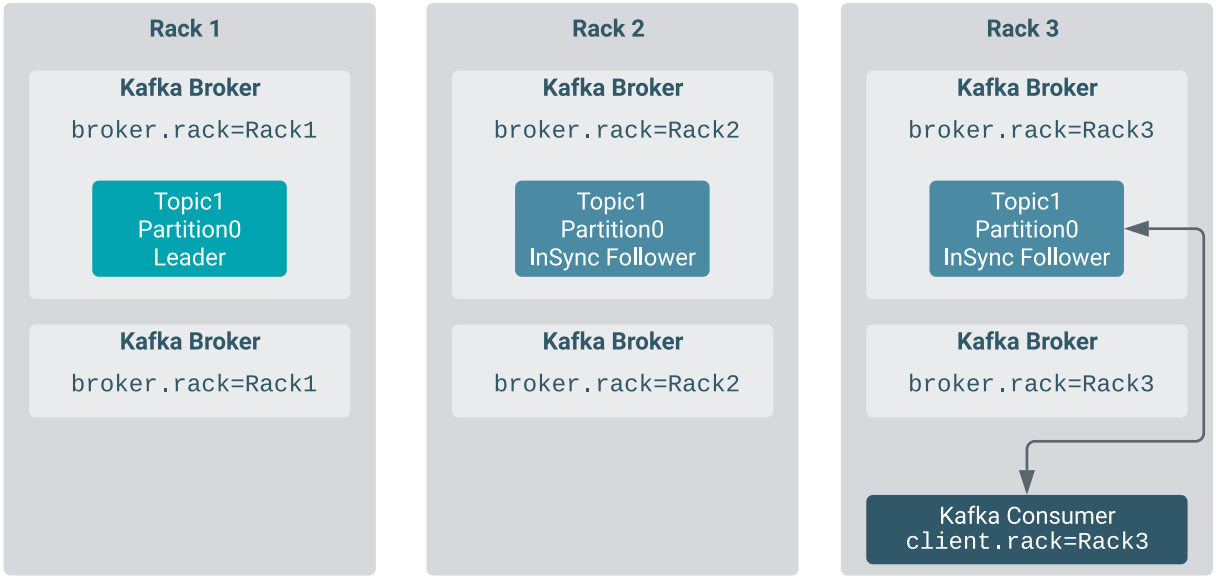
Learn about leader fetching, which can be used to make Kafka consumers rack aware

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.

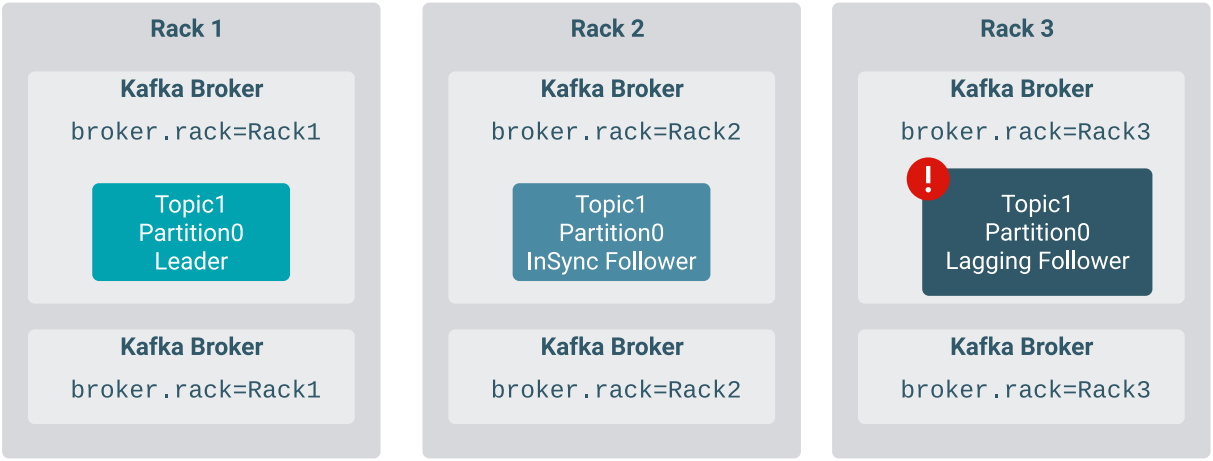
Compared to brokers or consumers, there are no producer specific rack-awareness features or toggles that you can enable. However, in a deployment where rack awareness is an important factor, you can make configuration changes so that producers make use of rack awareness and have messages replicated to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

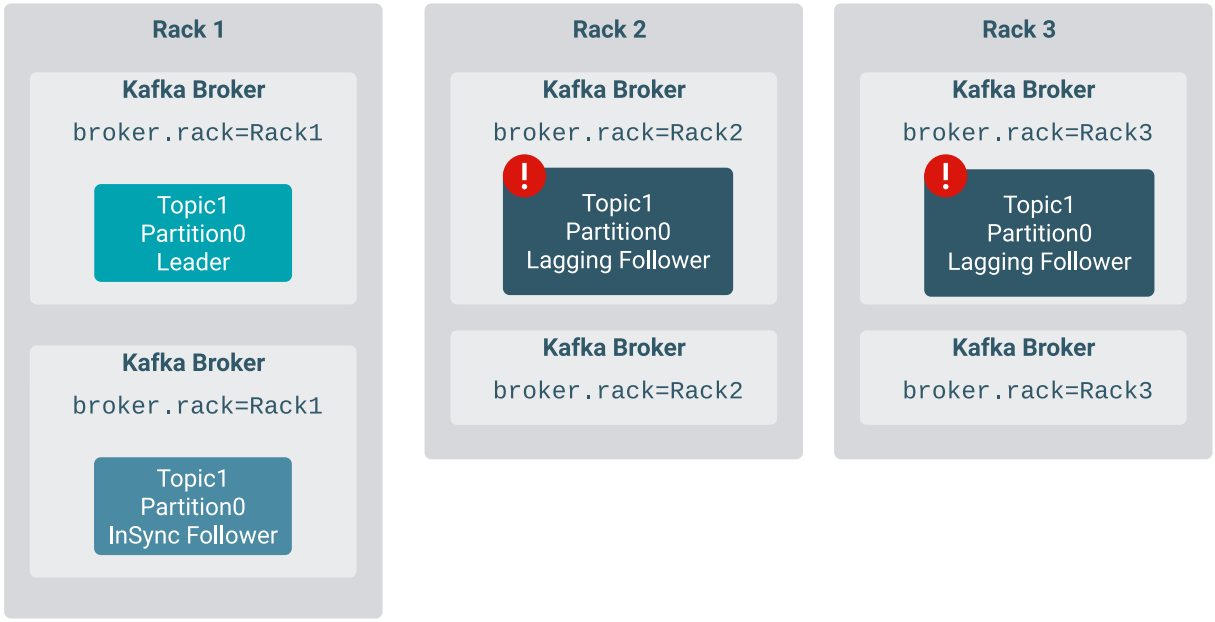


Understand however, that setting min.insync.replicas to 2 does not universally work for all deployments and may not guarantee that you always have your produced message in at least two racks. Configuration depends on the number of replicas, as well as the number of racks and brokers.

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.