

Managing Apache Kafka

Date published: 2019-12-18

Date modified: 2020-08-10



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

Management basics.....	4
Broker log management.....	4
Record management.....	4
Broker garbage log collection and log rotation.....	5
Client and broker compatibility across Kafka versions.....	5
Managing topics across multiple Kafka clusters.....	6
Set up MirrorMaker in Cloudera Manager.....	7
Settings to avoid data loss.....	8
Broker migration.....	8
Migrate brokers by modifying broker IDs in meta.properties.....	8
Use rsync to copy files from one broker to another.....	9
Disk management.....	10
Monitoring.....	10
Handling disk failures.....	10
Disk Replacement.....	10
Disk Removal.....	11
Reassigning replicas between log directories.....	11
Retrieving log directory replica assignment information.....	12
Metrics.....	12
Building Cloudera Manager charts with Kafka metrics.....	13
Essential metrics to monitor.....	14
Command Line Tools.....	16
Unsupported command line tools.....	16
kafka-topics.....	17
kafka-configs.....	18
kafka-console-producer.....	18
kafka-console-consumer.....	19
kafka-consumer-groups.....	20
kafka-reassign-partitions.....	22
Tool usage.....	23
Reassignment examples.....	25
kafka-log-dirs.....	26
zookeeper-security-migration.....	27
kafka-delegation-tokens.....	27
kafka-*-perf-test.....	28
Configuring log levels for command line tools.....	29
Understanding the kafka-run-class Bash Script.....	29

Management basics

Broker log management

Learn more about the configuration properties related to broker log management.

Kafka brokers save their data as log segments in a directory. The logs are rotated depending on the size and time settings.

The most common log retention settings to adjust for your cluster are shown below. These are accessible in Cloudera Manager via the Kafka Configuration tab.

- `log.dirs`: The location for the Kafka data (that is, topic directories and log segments).
- `log.retention.{ms|minutes|hours}`: The retention period for the entire log. Any older log segments are removed.
- `log.retention.bytes`: The retention size for the entire log.

In addition, there are many more configuration properties available for fine-tuning broker log management. For more information look for the following properties in the Broker Configs section in the upstream Apache Kafka documentation.

- `log.dirs`
- `log.flush.*`
- `log.retention.*`
- `log.roll.*`
- `log.segment.*`

Related Information

[Broker Configs](#)

Record management

Learn how you can manage records.

There are two pieces to record management, log segments and log cleaner.

As part of the general data storage, Kafka rolls logs periodically based on size or time limits. Once either limit is hit, a new log segment is created with the all new data being placed there, while older log segments should generally no longer change. This helps limit the risk of data loss or corruption to a single segment instead of the entire log.

- `log.roll.{ms|hours}`: The time period for each log segment. Once the current segment is older than this value, it goes through log segment rotation.
- `log.segment.bytes`: The maximum size for a single log segment.

There is an alternative to simply removing log segments for a partition. There is another feature based on the log cleaner. When the log cleaner is enabled, individual records in older log segments can be managed differently:

- `log.cleaner.enable`: This is a global setting in Kafka to enable the log cleaner.
- `cleanup.policy`: This is a per-topic property that is usually set at topic creation time. There are two valid values for this property, delete and compact.
- `log.cleaner.min.compaction.lag.ms`: This is the retention period for the “head” of the log. Only records outside of this retention period will be compacted by the log cleaner.

The compact policy, also called log compaction, assumes that the “most recent Kafka record is important.” Some examples include tracking a current email address or tracking a current mailing address. With log compaction, older records with the same key are removed from a log segment and the latest one is kept. This effectively removes some offsets from the partition.

Broker garbage log collection and log rotation

Learn more about broker garbage collection and how garbage log rotation can be configured.

About this task

Both broker JVM garbage collection and JVM garbage log rotation is enabled by default in the Kafka version delivered with Runtime. Garbage collection logs are written in the agent process directory by default.

Example path:

```
/run/cloudera-scm-agent/process/99-kafka-KAFKA_BROKER/kafkaServer-gc.log
```

Changing the default directory of garbage collection logs is currently not supported. However, you can configure properties related garbage log rotation.

Garbage log rotation properties can be configured with the Kafka Broker Environment Advanced Configuration Snippet (Safety Valve) property.

Procedure

1. In Cloudera Manager, go to the Kafka service and click Configuration.
2. Find the Kafka Broker Environment Advanced Configuration Snippet (Safety Valve) property.
3. Add the following line to the property:

Modify the values of as required.

```
KAFKA_GC_LOG_OPTS="-XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=100M"
```

The flags used are as follows:

- `+UseGCLogFileRotation`: Enables garbage log rotation.
 - `-XX:NumberOfGCLogFiles`: Specifies the number of files to use when rotating logs.
 - `-XX:GCLogFileSize`: Specifies the size when the log will be rotated.
4. Click on Save Changes.
 5. Restart the Kafka service to apply the changes.

Results

Kafka garbage log rotation is configured.

Client and broker compatibility across Kafka versions

An overview on client and broker version compatibility.

Maintaining compatibility across different Kafka clients and brokers is a common issue. Mismatches among client and broker versions can occur as part of any of the following scenarios:

- Upgrading your Kafka cluster without upgrading your Kafka clients.
- Using a third-party application that produces to or consumes from your Kafka cluster.
- Having a client program communicating with two or more Kafka clusters (such as consuming from one cluster and producing to a different cluster).
- Using Flume or Spark as a Kafka consumer.

In these cases, it is important to understand client/broker compatibility across Kafka versions. Here are general rules that apply:

- Newer Kafka brokers can talk to older Kafka clients. The reverse is not true: older Kafka brokers cannot talk to newer Kafka clients.
- Changes in either the major part or the minor part of the upstream version *major.minor* determines whether the client and broker are compatible. Differences among the maintenance versions are not considered when determining compatibility.

As a result, the general pattern for upgrading Kafka from version A to version B is:

1. Change Kafka server.properties to refer to version A.
2. Upgrade the brokers to version B and restart the cluster.
3. Upgrade the clients to version B.
4. After all the clients are upgraded, change the properties from the first step to version B and restart the cluster.

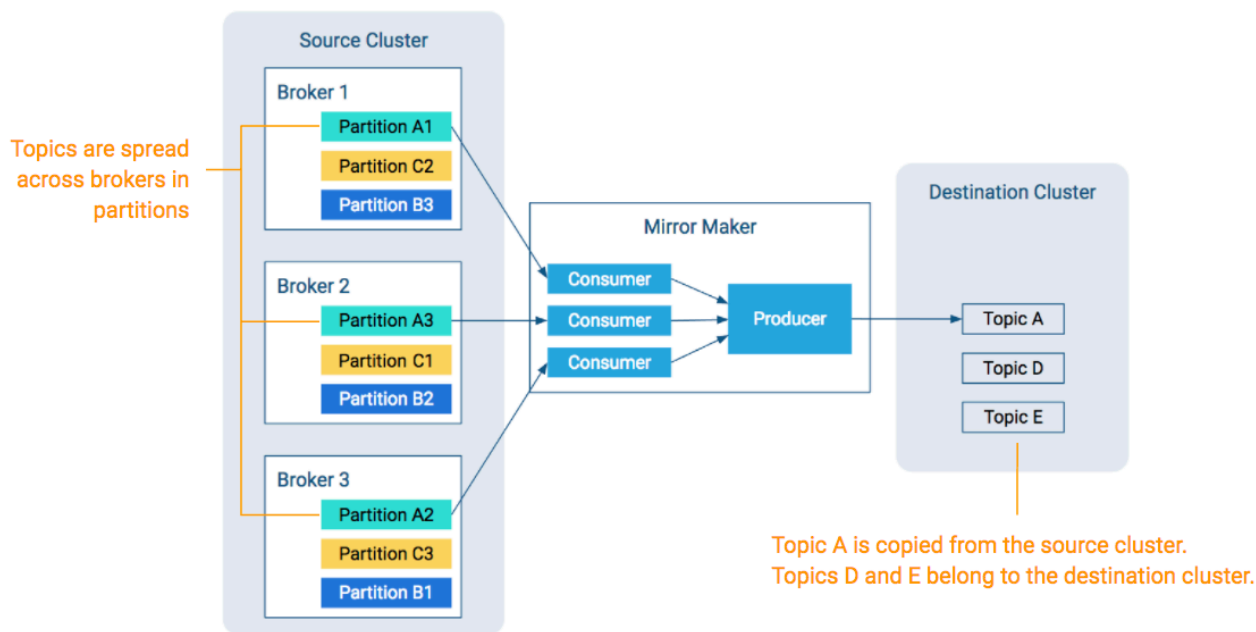
Managing topics across multiple Kafka clusters

You may have more than one Kafka cluster to support:

- Geographic distribution
- Disaster recovery
- Organizational requirements

You can distribute messages across multiple clusters. It can be handy to have a copy of one or more topics from other Kafka clusters available to a client on one cluster. Mirror Maker is a tool that comes bundled with Kafka to help automate the process of mirroring or publishing messages from one cluster to another. "Mirroring" occurs between clusters where "replication" distributes message within a cluster.

Figure 1: Mirror Maker Makes Topics Available on Multiple Clusters



While the diagram shows copying to one topic, Mirror Maker's main mode of operation is running continuously, copying one or more topics from the source cluster to the destination cluster.

Keep in mind the following design notes when configuring Mirror Maker:

- Mirror Maker runs as a single process.
- Mirror Maker can run with multiple consumers that read from multiple partitions in the source cluster.

- Mirror Maker uses a single producer to copy messages to the matching topic in the destination cluster.

Set up MirrorMaker in Cloudera Manager

Learn how to set up MirrorMaker in Cloudera Manager

Before you begin

Consider the following before setting up MirrorMaker for Kafka:

Consumer/Producer Compatibility

The Mirror Maker consumer needs to be client compatible with the source cluster. The Mirror Maker producer needs to be client compatible with the destination cluster. See [Client and Broker Compatibility Across Kafka Versions](#) for further details on compatibility.

Topic Differences between Clusters

Because messages are copied from the source cluster to the destination cluster—potentially through many consumers funneling into a single producer—there is no guarantee of having identical offsets or timestamps between the two clusters. In addition, as these copies occur over the network, there can be some mismatching due to retries or dropped messages.

Optimize Mirror Maker Producer Location

Because Mirror Maker uses a single producer and since producers typically have more difficulty with high latency and/or unreliable connections, it is better to have the producer run “closer” to the destination cluster, meaning in the same data center or on the same rack.

Destination Cluster Configuration

Before starting Mirror Maker, make sure that the destination cluster is configured correctly:

- Make sure there is sufficient disk space to copy the topic from the source cluster to the destination cluster.
- Make sure the topic exists in the destination cluster or enable topic auto creation by setting the `auto.create.topics.enable` property to true. The latter can be done in Cloudera Manager by going to `KafkaConfiguration`.

Kerberos and Mirror Maker

As mentioned earlier, Mirror Maker runs as a single process. The resulting consumers and producers rely on a single configuration setup. Mirror Maker requires that the source cluster and the destination cluster belong to the same Kerberos realm.

Procedure

1. In Cloudera Manager, select the Kafka service.
2. Choose Action Add Role Instances.
3. Under Kafka Mirror Maker, click Select hosts.
4. Select the host where Mirror Maker will run and click Continue.
5. Fill in the Destination Broker List and Source Broker List with your source and destination Kafka clusters.
Use host name, IP address, or fully qualified domain name.
6. Fill out the Topic Whitelist.
The allowlist (whitelist) is required.
7. Fill out the TLS/SSL sections if security needs to be enabled.
8. Start the Mirror Maker instance.

Results

MirrorMaker is configured and started.

Related Information

[Client and broker compatibility across Kafka versions](#)

Settings to avoid data loss

An overview of default configuration properties to avoid data loss.

If for some reason the producer cannot deliver messages that have been consumed and committed by the consumer, it is possible for a MirrorMaker process to lose data. The following collection of properties are set by default and help with preventing data loss when replicating data. In addition, note that MirrorMaker starts correctly if you enter the numeric values in the configuration snippet (rather than using "max integer" for retries and "max long" for max.block.ms).

Producer settings

- `acks=all`
- `retries=2147483647`
- `max.block.ms=9223372036854775807`

Consumer setting

- `auto.commit.enable=false`

MirrorMaker setting

- `abort.on.send.failure=true`

Broker migration

Learn more about the options you have when migrating brokers and copying data between brokers.

In case of catastrophic hardware failure brokers can be moved to a new host in a Kafka cluster. There are multiple methods with which you can move brokers between clusters:

Using `kafka-reassign-partitions` tool

This method involves more manual work to modify JSON, but does not require manual edits to configuration files.

Modify the broker IDs in `meta.properties`

This method involves less manual work, but requires modifying an internal configuration file.

Using `rsync`

You can use `rsync` to copy over all data from an old broker to a new broker, preserving modification times and permissions.

Continue reading for instructions on how to move brokers by modifying broker IDs or using `rsync`. For instructions on how to migrate brokers with `kafka-reassign-partitions`, see the `kafka-reassign-partitions` tool description. For specific instructions on how to migrate existing Kafka partitions to JBOD configured disks, see [JBOD disk migration](#).

Related Information

[kafka-reassign-partitions](#)

[JBOD disk migration](#)

Migrate brokers by modifying broker IDs in `meta.properties`

Learn how to migrate brokers between hosts in a Kafka cluster by modifying broker IDs in `meta.properties`.

About this task

In case of catastrophic hardware failure brokers can be moved to a new host in a Kafka cluster. The following steps walk you through broker migration by modifying broker IDs in `meta.properties`. Compared to migration with the help of the `kafka-reassign-partitions` tool, this method involves less manual work, but requires modifying an internal configuration file.



Important: Data intensive administration operations such as rebalancing partitions, adding a broker, removing a broker, or bootstrapping a new machine can cause significant additional load on the cluster. Therefore, Cloudera highly recommends that you migrate one broker at a time and migrate brokers when there is minimal load on the cluster

Before you begin

- Verify that the cluster is healthy.
- Verify that all replicas are in sync.

Procedure

1. Start up the new broker as a member of the old cluster.
This creates files in the data directory.
2. Stop both the new broker and the old broker that it is replacing.
3. Change `broker.id` of the new broker to the `broker.id` of the old one both in Cloudera Manager and in `data directory/meta.properties`.
4. Optional: Run `rsync` to copy files from one broker to another.
5. Start up the new broker.
When started the new broker re-replicates data from the other nodes.

Results

The Kafka broker is migrated to a new host.

Use `rsync` to copy files from one broker to another

Learn how to use `rsync` to copy over all data from an old broker to a new broker.

About this task

You can run `rsync` command to copy over all data from an old broker to a new broker, preserving modification times and permissions. Using `rsync` allows you to avoid having to re-replicate the data from the leader.

Before you begin

- Ensure that the disk structures match between the two brokers, or verify the `meta.properties` file between the source and destination brokers (because there is one `meta.properties` file for each data directory).

Procedure

Run the following command on destination broker:

```
rsync -avz  
    src_broker:src_data_dir  
    dest_data_dir
```

If you plan to change the broker ID, edit `dest_data_dir/meta.properties`.

Results

Data from the source broker is copied over to the destination broker.

Disk management

Monitoring

Recommendations on what to monitor on clusters.

Cloudera recommends that administrators continuously monitor the following on a cluster:

Replication Status

Monitor replication status using Cloudera Manager Health Tests. Cloudera Manager automatically and continuously monitors both the `OfflineLogDirectoryCount` and `OfflineReplicaCount` metrics. Alerts are raised when failures are detected.

Disk Capacity

Monitor free space on mounted disks and open file descriptors. Reassign partitions or move log files around if necessary. For more information, see the `kafka-reassign-partitions` tool description.

Related Information

[Cloudera Manager Health Tests](#)

[kafka-reassign-partitions](#)

Handling disk failures

An overview on how to handle disk failures.

Cloudera Manager has built in monitoring functionalities that automatically trigger alerts when disk failures are detected. When a log directory fails, Kafka also detects the failure and takes the partitions stored in that directory offline.



Important: If there are no healthy log directories present in the system, the broker stops working.

The cause of disk failures can be analyzed with the help of the `kafka-log-dirs` tool, or by reviewing the error messages of `KafkaStorageException` entries in the Kafka broker log file. To access the log file go to `InstancesLog FilesRole Log File`.

In case of a disk failure, a Kafka administrator can carry out either of the following actions. The action taken depends on the failure type and system environment:

- Replace the faulty disk with a new one.
- Remove the disk and redistribute data across remaining disks to restore the desired replication factor.



Note: Disk replacement and disk removal both require stopping the broker. Therefore, Cloudera recommends that you perform these actions during a maintenance window.

Related Information

[kafka-log-dirs](#)

Disk Replacement

Learn how to replace a disk.

About this task

In case of a disk failure, a Kafka administrator can replace the faulty disk with a new one.

Procedure

1. Stop the broker that has a faulty disk:
 - a) In Cloudera Manager, go to the Kafka service, select Instances and select the broker.
 - b) Go to ActionsGracefully stop this Kafka Broker.
2. Replace the disk.
3. Mount the disk.
4. Set up the directory structure on the new disk the same way as it was set up on the previous disk.



Note: You can find the directory paths for the old disk in the Data Directories property of the broker.

5. Start the broker:
 - a) In Cloudera Manager go to the Kafka service, selectInstances and select the broker.
 - b) Go to ActionsStart this Kafka Broker.

Results

The disk is replaced. The Kafka broker re-creates topic partitions in the same directory by replicating data from other brokers.

Disk Removal

Learn how to remove a disk from the configuration.

About this task

In case of a disk failure, a Kafka administrator can remove the disk and redistribute data across remaining disks to restore the desired replication factor.

Procedure

1. Stop the broker that has a faulty disk:
 - a) In Cloudera Manager, go to the Kafka service, select Instances and select the broker.
 - b) Go to ActionsGracefully stop this Kafka Broker.
2. Remove the log directories on the faulty disk from the broker:
 - a) Go to Configuration and find the Data Directories property.
 - b) Remove the affected log directories with the Remove button.
 - c) Enter a Reason for change, and then click Save Changes to commit the changes.
3. Start the broker:
 - a) In Cloudera Manager, go to the Kafka service, select Instances and select the broker.
 - b) Go to ActionsStart this Kafka Broker.

Results

The disk is removed from the configuration. The Kafka broker redistributes data across the cluster.

Reassigning replicas between log directories

Learn about replica reassignment between log directories.

Reassigning replicas between log directories can prove useful when you have multiple disks available, but one or more of them is nearing capacity. Moving a replica from one disk to another ensures that the service will not go down due to disks reaching capacity. To balance storage loads, the Kafka administrator has to continuously monitor the

system and reassign replicas between log directories on the same broker or across different brokers. These actions can be carried out with the `kafka-reassign-partitions` tool.

For more information on tool usage, see the `kafka-reassign-partitions` tool description.

Related Information

[kafka-reassign-partitions](#)

Retrieving log directory replica assignment information

Learn how log directory replica assignment information can be retrieved.

To optimize replica assignment across log directories, the list of partitions per log directory and the size of each partition is required. This information can be exposed with the `kafka-log-dirs` tool.

For more information on tool usage, see the `kafka-log-dirs` tool description.

Related Information

[kafka-log-dirs](#)

Metrics

Learn about Kafka metrics and how to view them.

Kafka uses Yammer metrics to record internal performance measurements. The metrics are exposed via Java Management Extensions (JMX) and can be read with a JMX console.

Metrics Categories

There are metrics available in the various components of Kafka. In addition, there are some metrics specific to how Cloudera Manager and Kafka interact. This table has pointers to both the Apache Kafka metrics names and the Cloudera Manager metric names.

Table 1: Metrics by Category

Category	Cloudera Manager Metrics Doc	Apache Kafka Metrics Doc
Cloudera Manager Kafka Service	Base Metrics	
Broker	Broker Metrics , Broker Topic Metrics , Replica Metrics , Broker Topic Partition Metrics	Broker
Common Producer/Consumer		Client Client-to-Broker
Producer	Producer Metrics	Producer Producer Sender
Consumer	Consumer Metrics , Consumer Group Metrics	Consumer Group Consumer Fetch
Mirror Maker	Mirror Maker Metrics	Same as Producer or Consumer tables

Note the following about Cloudera Manager metrics:

Broker Topic Partitions Metrics:

Required for Streams Messaging Manager. These metrics have to be specifically enabled within the host's configuration.

Producer Metrics:

Required for Streams Messaging Manager. These metrics have to be specifically enabled within Kafka's configuration and are different from the metrics available on client instances.

Consumer and Consumer Groups Metrics

Generated by and required for Streams Messaging Manager. These metrics are different from the metrics available on client instances.

Viewing Metrics

Cloudera Manager records most of these metrics and makes them available via Chart Builder.

Because Cloudera Manager cannot track metrics on any clients (that is, producer or consumer), you may wish to use an alternative JMX console program to check metrics. There are several JMX console options, for example:

- JConsole, which comes bundled with the JDK.
- VisualVM, with the MBeans plugin.

Related Information

[JConsole Documentation](#)

[VisualVM Homepage](#)

Building Cloudera Manager charts with Kafka metrics

A collection query examples to build Kafka metrics charts within Cloudera Manager.

Cloudera Manager enables you to build charts based on Kafka specific metrics. To access the chart builder go to ChartsChart Builderin Cloudera Manager. The following are a few specific examples of queries for cloudera metrics:

Controllers across all brokers

This chart shows the active controller across all brokers. It is useful for checking active controller status (should be one at any given time, transitions should be fast).

```
SELECT
  kafka_active_controller
WHERE
  roleType=KAFKA_BROKER
```

Network idle rate

>Chart showing the network processor idle rate across all brokers. If idle time is always zero, then probably the num.network.threads property may need to be increased.

```
SELECT
  kafka_network_processor_avg_idle_rate
WHERE
  roleType=KAFKA_BROKER
```

Partitions per broker

Chart showing the number of partitions per broker. It is useful for detecting partition imbalances early.

```
SELECT
  kafka_partitions
WHERE
  roleType=KAFKA_BROKER
```

Partition activity

Chart tracking partition activity on a single broker.

```
SELECT
  kafka_partitions, kafka_under_replicated_partitions
WHERE
  hostname=host1.domain.com
```

Mirror Maker activity

Chart for tracking Mirror Maker behavior. Since Mirror Maker has one or more consumers and a single producer, most consumer or metrics should be usable with this query.

```
SELECT
  producer or consumer metric
WHERE
  roleType=KAFKA_MIRROR_MAKER
```

Related Information

[Charting Time-Series Data](#)

Essential metrics to monitor

Cloudera Manager collects a high number of performance metrics for the Kafka services running on your clusters. Certain metrics should be monitored in any Kafka deployment as they can help you to improve the stability and performance of your Kafka deployment.

The following tables collect the Kafka broker metrics that Cloudera recommends you to monitor in any Kafka deployment. For more information on metrics, including a full list of Kafka metrics, see [Cloudera Manager Metrics](#).

Table 2: ZooKeeper connectivity metrics

Metric Name	Description	Unit	Importance	Parents	Version Availability
kafka_zookeeper_expires_rate	Measures the session expires per second.	Expires per second	High	cluster, kafka, rack	CDH 5, CDH 6, CDP 7
kafka_zookeeper_request_latency_avg	Request latency between the broker and Zookeeper.	ms	High	cluster, kafka, rack	CDH 5, CDH 6, CDP 7

Table 3: Active controller metrics

Metric Name	Description	Unit	Importance	Parents	Version Availability
kafka_active_controller	Shows the number of active controllers at a given time. Ideally it should be 1.	Number of controllers	High	cluster, kafka, rack	CDH 5, CDH 6, CDP 7

Table 4: Network metrics

Metric Name	Description	Unit	Importance	Parents	Version Availability
kafka_network_processor_avg_idle	The average free capacity of the network processors. Should be > 0.3.	Percentage of free capacity	High	cluster, kafka, rack	CDH 5, CDH 6, CDP 7

Metric Name	Description	Unit	Importance	Parents	Version Availability
kafka_request_queue_size	Size of the request queue in Kafka.	Message count	Medium	cluster, kafka, rack	CDH 5, CDH 6, CDP 7
kafka_response_queue_size	Size of the response queue in Kafka.	Message count	Medium	cluster, kafka, rack	CDH 5, CDH 6, CDP 7
kafka_messages_received_rate	Number of messages written to topic on this broker.	Messages per second	Medium	cluster, kafka, rack	CDH 5, CDH 6, CDP 7

Table 5: Disk utilization metrics

Metric Name	Description	Unit	Importance	Parents	Version Availability
kafka_produce_local_time_rate	Local Time spent in responding to Produce requests.	Requests per second	Low	cluster, kafka, rack	CDH 5, CDH 6, CDP 7
kafka_log_flush_rate	Rate of flushing Kafka logs to disk.	Flushes per second	Low	cluster, kafka, rack	CDH 5, CDH 6, CDP 7
kafka_request_handler_avg_idle_rate	The average free capacity of the request handler. Should be > 0.3.	Percentage of free capacity	High	cluster, kafka, rack	CDH 5, CDH 6, CDP 7



Note: While the disk utilization metrics do not measure disk performance directly, they can point to problems with the disks.

Table 6: Kafka metrics

Metric Name	Description	Unit	Importance	Parents	Version Availability
kafka_broker_state	The state the broker is in. 0 = NotRunning, 1 = Starting, 2 = RecoveringFromUncleanShutdown, 3 = RunningAsBroker, 4 = RunningAsController, 6 = PendingControlledShutdown, 7 = BrokerShuttingDown	Discrete states	Medium	cluster, kafka, rack	CDH 5, CDH 6, CDP 7
kafka_jvm_gc_runs_rate	Number of garbage collector runs performed on this broker.	Events per second	Medium	cluster, kafka, rack	CDH 5, CDH 6, CDP 7
kafka_isr_expands_rate	ISR expands per second.	Events per second	Medium	cluster, kafka, rack	CDH 5, CDH 6, CDP 7
kafka_isr_shrinks_rate	ISR shrinks per second.	Events per second	Medium	cluster, kafka, rack	CDH 5, CDH 6, CDP 7
kafka_max_replication_lag	Maximum replication lag on the broker, across all fetchers, topics, and partitions.	Messages	Medium	cluster, kafka, rack	CDH 5, CDH 6, CDP 7
kafka_offline_partitions	Number of offline partitions.	Partition count	High	cluster, kafka, rack	CDH 5, CDH 6, CDP 7

Metric Name	Description	Unit	Importance	Parents	Version Availability
kafka_under_min_isr_partition_count	Count of partitions with less than the configured minimum in-sync replicas available.	Partition count	High	cluster, kafka, rack	CDH 5, CDH 6, CDP 7
kafka_under_replicated_partitions	Count of partitions with unavailable replicas.	Partition count	Low	cluster, kafka, rack	CDH 5, CDH 6, CDP 7

Related Information

[Cloudera Manager Metrics](#)

Command Line Tools

Kafka command line tools overview.

In some situations, it is convenient to use the command line tools available in Kafka to administer your cluster. However, it is important to note that not all tools available for Kafka are supported by Cloudera. Moreover, certain administration tasks can be carried out more easily and conveniently using Cloudera Manager.

When administering Kafka with command line tools, be aware of the following:

- Use Cloudera Manager to start and stop Kafka and Zookeeper services. Do not use the `kafka-server-start`, `kafka-server-stop`, `zookeeper-server-start`, or `zookeeper-server-stop` commands.
- For a parcel installation, all Kafka command line tools are located in `/opt/cloudera/parcels/KAFKA/lib/kafka/bin/`. For a package installation, all such tools can be found in `/usr/bin/`.
- Ensure that the `JAVA_HOME` environment variable is set to your JDK installation directory before using the command-line tools. For example:

```
export JAVA_HOME=/usr/java/jdk1.8.0_144-cloudera
```

- Using any Zookeeper command manually can be very difficult to get right when it comes to interaction with Kafka. Cloudera recommends that you avoid doing any write operations or ACL modifications in Zookeeper.



Note: Output examples in the command line tool descriptions are cleaned and formatted for easier readability.

Unsupported command line tools

Kafka command line tools not supported by Cloudera.

The following tools can be found as part of the Kafka distribution, but their use is generally discouraged for various reasons as documented here.

Tool	Notes
connect-mirror-maker	Use Streams Replication Manager instead.
connect-distributed connect-standalone	Use Cloudera Manager to deploy and manage Kafka Connect workers. In CDP Kafka Connect workers are represented by Kafka Connect roles, which can be deployed under a Kafka service.
kafka-acls	Cloudera recommends using Ranger for authorization instead.
kafka-broker-api-versions	Primarily useful for Client-to-Broker protocol related development.

Tool	Notes
kafka-configs	Use Cloudera Manager to adjust any broker or security properties instead of the kafka#configs tool. This tool should only be used to modify topic properties.
kafka-delete-records	Cloudera does not recommend using this tool in CDP.
kafka-mirror-maker	Use Cloudera Manager to deploy Mirror Maker instances.
kafka-preferred-replica-election	This tool causes leadership for each partition to be transferred back to the 'preferred replica'. It can be used to balance leadership among the servers. This tool is deprecated, Cloudera recommends that you use kafka-leader-election, kafka-reassign-partitions, or Cruise Control instead.
kafka-replay-log-producer	Can be used to “rename” a topic.
kafka-replica-verification	Validates that all replicas for a set of topics have the same data. This tool is a “heavy duty” version of the ISR column of kafka-topics tool.
kafka-server-start kafka-server-stop	Use Cloudera Manager to manage any Kafka host.
kafka-verifiable-consumer kafka-verifiable-producer	These scripts are intended for system testing.
zookeeper-server-start zookeeper-server-stop	Use Cloudera Manager to manage any Zookeeper host.
zookeeper-shell	Limit usage of this script to reading information from Zookeeper.

kafka-topics

Learn more about the kafka-topics tool.

Use the kafka-topics tool to generate a snapshot of topics in the Kafka cluster.

```
kafka-topics --bootstrap-server [BROKER_HOST]:[PORT] --describe
```

```
Topic: topic-a1      PartitionCount:3      ReplicationFactor:3      Configs:
      Topic: topic-a1      Partition: 0      Leader: 64      Replicas: 6
4,62,63      Isr: 64,62,63
      Topic: topic-a1      Partition: 1      Leader: 62      Replicas:
62,63,64      Isr: 62,63,64
      Topic: topic-a1      Partition: 2      Leader: 63      Replicas: 6
3,64,62      Isr: 63,64,62
Topic: topic-a2      PartitionCount:1      ReplicationFactor:3      Configs:
      Topic: topic-a2      Partition: 0      Leader: 64      Replicas: 64
,62,63      Isr: 64,62,63
```

The output lists each topic and basic partition information. Note the following about the output:

- Partition count: The more partitions, the higher the possible parallelism among consumers and producers.
- Replication factor: Shows 1 for no redundancy and higher for more redundancy.
- Replicas and in-sync replicas (ISR): Shows which broker ID's have the partitions and which replicas are current.

There are situations where this tool shows an invalid value for the leader broker ID or the number of ISRs is fewer than the number of replicas. In those cases, there may be something wrong with those specific topics.

It is possible to change topic configuration properties using this tool. Increasing the partition count, the replication factor or both is not recommended.

kafka-configs

Learn how to use the kafka-configs tool to set check or uncheck topic properties.

The kafka-configs tool allows you to set and unset properties to topics. Cloudera recommends that you use Cloudera Manager instead of this tool to change properties on brokers, because this tool bypasses any Cloudera Manager safety checks.

Setting a topic property:

```
kafka-configs --zookeeper [ZKHOST] --entity-type topics --entity-name [TOPIC] --alter --add-config [PROPERTY_NAME]=[VALUE]
```

Checking a topic property:

```
kafka-configs --zookeeper [ZKHOST] --entity-type topics --entity-name [TOPIC] --describe
```

Unsetting a topic property:

```
kafka-configs --zookeeper [ZKHOST] --entity-type topics --entity-name [TOPIC] --alter --delete-config [PROPERTY_NAME]
```

See the upstream Apache Kafka documentation for a complete list of available topic configuration properties.

Related Information

[Topic-Level Configs](#)

kafka-console-producer

Learn how you can use the kafka-console-producer tool to produce messages to a topic.

This tool is used to write messages to a topic in a text based format.

The following examples demonstrate the basic usage of the tool. In addition to reviewing these examples, you can also use the --help option to see a list of all available options.

Produce messages to a topic

To start producing message to a topic, you need to run the tool and specify a server as well as a topic.

```
kafka-console-producer --broker-list [HOST1:PORT1] --topic [TOPIC]
```

Start typing messages once the tool is running.

```
>my first message
>my second message
```

Alternatively, you can also produce the contents of a file to a topic.

```
cat [FILE] | kafka-console-producer --broker-list [HOST1:PORT1] --topic [TOPIC]
```

Produce messages to a topic in a secure cluster

In order to use the tool on a secure cluster, additional client configuration is required. You do this by creating a .properties file that contains the necessary configurations to run the tool on a secure cluster. Which properties are configured in this file depends on the security configuration of your cluster. See the client configuration topics in

Securing Apache Kafka to learn more about which exact properties you need to use for your security configuration. Once the file is created, you can use `--producer.config` to pass the file to the tool.

Example `.properties` file:

```
security.protocol = SASL_SSL
sasl.mechanism=GSSAPI
sasl.kerberos.service.name = kafka
ssl.truststore.location = /var/private/ssl/kafka.client.truststore.jks
ssl.truststore.password = test1234
sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required useTicketCache=true;
```

This example shows what properties you have to set when both Kerberos and TLS/SSL are configured.

Do the following to run the tool in a secure cluster:

1. Create a `.properties` file.

Use the example above. Make changes as necessary.

2. Run the tool with the `--producer.config` option.

```
kafka-console-producer --broker-list [HOST1:PORT1] --topic [TOPIC] --producer.config client.properties
```

Define a key-value delimiter

It is possible to define a key-value delimiter for the given producer instance. The delimiter can vary each time you run the tool. For example, if you run the tool with the delimiter set to `-` and then a second time using `:`, Kafka will know how to store the data. The first term is always stored as the key, the second as the value. As a result, when the data written this way is consumed, the keys and values are consumed irrespective of what delimiter was used when the records were written to the topic.

To produce messages with key-value delimiters, you need to set two properties, `parse.key` and `key.separator`. Both properties can be set with the `--properties` option.

```
kafka-console-producer --broker-list [HOST1:PORT1] --topic [TOPIC] --property parse.key=true --property key.separator=":"
```

Configure retry backoff

Before each retry, the producer refreshes the metadata of the relevant topics. You can configure the amount of time the producer waits before refreshing the metadata with the `--retry-backoff-ms` option.

```
kafka-console-producer --broker-list [HOST1:PORT1] --topic [TOPIC] --retry-backoff-ms 1000
```

Related Information

[Configure Kafka clients for TLS/SSL encryption](#)

[Enable Kerberos authentication](#)

[Client authentication using delegation tokens](#)

[Configure Kafka clients for LDAP authentication](#)

[Configure Kafka clients for PAM authentication](#)

kafka-console-consumer

Learn how to use the `kafka-console-consumer` tool.

The kafka-console-consumer tool can be useful in a couple of ways:

- Acting as an independent consumer of particular topics. This can be useful to compare results against a consumer program that you've written.
- To test general topic consumption without the need to write any consumer code.

Examples of usage:

```
kafka-console-consumer --bootstrap-server [BROKER1],[BROKER2]... --to
pic [TOPIC] --from-beginning
[record-earliest-offset]
[record-earliest-offset+1]
```

Note the following about the tool:

- This tool prints all records and keeps outputting as more records are written to the topic.
- If the kafka-console-consumer tool is given no flags, it displays the full help message.
- In older versions of Kafka, it may have been necessary to use the --new-consumer flag. As of Apache Kafka version 0.10.2, this is no longer necessary.

kafka-consumer-groups

Learn how to use the kafka-consumer-groups tool.

The kafka-consumer-groups tool can be used to list all consumer groups, describe a consumer group, delete consumer group info, or reset consumer group offsets. The following topic gives an overview on how to describe or reset consumer group offsets.

Describe Offsets

This tool is primarily used for describing consumer groups and debugging any consumer offset issues, like consumer lag. The output from the tool shows the log and consumer offsets for each partition connected to the consumer group that is being described. You can see at a glance which consumers are current with their partition and which ones are lagging. From there, you can determine which partitions (and likely the corresponding brokers) are slow.

Using the tool on secure and unsecure clusters differs slightly. On secure clusters, you have to use the command-config option together with an appropriate property file.

Describing offsets on an unsecure cluster

Use the following command to describe offsets committed to Kafka:

```
kafka-consumer-groups --bootstrap-server broker01.example.com:909
2 --describe --group flume
```

Output Example:

GROUP	TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG
OWNER					
flume	t1	0	1	3	2
test-consumer-group_postamac.local-1456198719410-29ccd54f-0					

Describing offsets on a secure cluster

In order to describe offsets on a secure Kafka cluster, the consumer-groups tool has to be run with the command-config option. The command-config option specifies the property file that contains the necessary configurations to run the tool on a secure cluster. Which properties are configured in this file depends on the security configuration of your cluster.

Example client.properties file:

```
exclude.internal.topics=false
security.protocol = SASL_SSL
sasl.kerberos.service.name = kafka
ssl.truststore.location = /var/private/ssl/kafka.client.truststore.jks
ssl.truststore.password = test1234
```

This example shows what properties you have to set when both Kerberos and TLS/SSL are configured.

To describe offsets do the following:

1. Pass the jaas.conf file location as a JVM parameter.

```
export KAFKA_OPTS='-Djava.security.auth.login.config=path/to/jaas.conf'
```

2. Create a client.properties file.

Use the example above. Make changes as necessary.

3. Run the tool with the command-config option.

```
kafka-consumer-groups --bootstrap-server broker01.example.com:9093 --describe --command-config client.properties --group flume
```

Resetting Offsets

You can use the `--reset-offset` option to reset the offsets of a consumer group to a particular value. The tool can be used to reset all offsets on all topics. However, this is something you probably won't ever want to do. Therefore, it is highly recommended that you exercise caution when resetting offsets.

To reset offsets you need to define a scope, an execution option, and a scenario.

Scope

There are two supported scopes:

- `--topic`: Restricts the change to a specific topic or a specific set of partitions within a topic.
- `--all-topics`: Executes the change for all topics.



Note: Choosing a scope is mandatory for all scenarios except for the `--from-file` scenario where specifying a scope is not required.

Execution option

There are three execution options:

- `--dry-run --reset-offsets`: Default option. Displays which offsets will be reset if the process is executed.
- `--execute --reset-offsets`: Executes the process.
- `--export --reset-offsets`: Exports results in CSV format.

Scenarios

There are a number of supported scenarios. Scenarios control what value the offsets are reset to. Specifying a scenario is mandatory.

- `--to-datetime`
- `--by-period`
- `--to-earliest`
- `--to-latest`

- --shift-by
- --from-file
- --to-current

Example:

```
kafka-consumer-groups --dry-run --reset-offsets --bootstrap-server broker-01.example.com:9092 --group flume --topic t1 --to-current
```

kafka-reassign-partitions

An overview of the kafka-reassign-partitions tool.

This tool provides substantial control over partitions in a Kafka cluster. It is mainly used to balance storage loads across brokers through the following reassignment actions:

- Change the ordering of the partition assignment list. Used to control leader imbalances between brokers.
- Reassign partitions from one broker to another. Used to expand existing clusters.
- Reassign partitions between log directories on the same broker. Used to resolve storage load imbalance among available disks in the broker.
- Reassign partitions between log directories across multiple brokers. Used to resolve storage load imbalance across multiple brokers.

The tool uses two JSON files for input. Both of these are created by the user. The two files are the following:

- Topics-to-Move JSON
- Reassignment Configuration JSON

Topics-to-Move JSON

This JSON file specifies the topics that you want to reassign. This is a simple file that tells the kafka-reassign-partitions tool which partitions it should look at when generating a proposal for the reassignment configuration. The user has to create the topics-to-move JSON file from scratch.

The format of the file is the following:

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

Reassignment Configuration JSON

This JSON file is a configuration file that contains the parameters used in the reassignment process. This file is created by the user, however, a proposal for its contents is generated by the tool. When the kafka-reassign-partitions tool is executed with the --generate option, it generates a proposed configuration which can be fine-tuned and saved as a JSON file. The file created this way is the reassignment configuration JSON. To generate a proposal, the tool requires a topics-to-move file as input.

The format of the file is the following:

```
{ "version": 1,
  "partitions": [
    { "topic": "mytopic1", "partition": 3, "replicas": [4,5], "log_dirs": [ "any", "any" ] },
    { "topic": "mytopic1", "partition": 1, "replicas": [5,4], "log_dirs": [ "any", "any" ] },
    { "topic": "mytopic2", "partition": 2, "replicas": [6,5], "log_dirs": [ "any", "any" ] }
  ]
}
```

The reassignment configuration contains multiple properties that each control and specify an aspect of the configuration. The Reassignment Configuration Properties table lists each property and its description.

Table 7: Reassignment Configuration Properties

Property	Description
topic	Specifies the topic.
partition	Specifies the partition.
replicas	Specifies the brokers that the selected partition is assigned to. The brokers are listed in order, which means that the first broker in the list is always the leader for that partition. Change the order of brokers to resolve any leader balancing issues among brokers. Change the broker IDs to reassign partitions to different brokers.
log_dirs	Specifies the log directory of the brokers. The log directories are listed in the same order as the brokers. By default any is specified as the log directory, which means that the broker is free to choose where it places the replica. By default, the current broker implementation selects the log directory using a round-robin algorithm. An absolute path beginning with a / can be used to explicitly set where to store the partition replica.

Notes and Recommendations:

- 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.
- This tool cannot be used to make an out-of-sync replica into the leader partition.
- Use this tool only when all brokers and topics are healthy.
- Anticipate system growth. Redistribute the load when the system is at 70% capacity. Waiting until redistribution becomes necessary due to reaching resource limits can make the redistribution process extremely time consuming.

Tool usage

Learn how to reassign partitions with the kafka-reassign-partitions-tool.

Procedure

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

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.

```
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. Review, and if required, modify the suggested reassignment configuration.
5. Save the file.
6. Start the redistribution process with the following command:

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



Note: Specifying a bootstrap server with the `--bootstrap-server` option is only required when an absolute log directory path is specified for a replica in the reassignment configuration JSON file.

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

Current partition replica assignment

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

Save this to use as the `--reassignment-json-file` option during rollback


```
Successfully started reassignment of partitions.
```

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

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

The tool prints the reassignment status of all partitions.

```
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

Partitions are reassigned.

Reassignment examples

A collection of examples that demonstrate how users can modify the proposed configuration file generated by the `kafka-reassign-partitions`.

There are multiple ways to modify the configuration file. The following list of examples shows how a user can modify a proposed configuration and what these changes do.

Suppose that the `kafka-reassign-partitions` tool generated the following proposed reassignment configuration:

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

Now let's look at how this reassignment configuration has to be changed in different reassignment scenarios.

Reassign partitions between brokers

To reassign partitions from one broker to another, change the broker ID specified in replicas. For example:

```
{ "topic": "mytopic1", "partition": 0, "replicas": [5, 2], "log_dirs": [ "any", "any" ] }
```

This reassignment configuration moves partition `mytopic1-0` from broker 1 to broker 5.

Reassign partitions to another log directory on the same broker

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": [1, 2], "log_dirs": [ "/log/directory1", "any" ] }
```

This reassignment configuration moves partition `mytopic1-0` to the `/log/directory1` log directory.

Reassign partitions between log directories across multiple brokers

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

```
{ "topic": "mytopic1", "partition": 0, "replicas": [5, 2], "log_dirs": [ "/log/directory1", "any" ] }
```

This reassignment configuration moves partition mytopic1-0 to /log/directory1 on broker 5.

Change partition assignment order (elect a new leader)

To change the ordering of the partition assignment list, change the order of the brokers in replicas. For example:

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

This reassignment configuration elects broker 2 as the new leader.

kafka-log-dirs

Learn how to use the kafka-log-dirs tool.

The kafka-log-dirs tool allows user to query a list of replicas per log directory on a broker. The tool provides information that is required for optimizing replica assignment across brokers.

To retrieve replica assignment information, run the following command:

```
kafka-log-dirs --describe --bootstrap-server hostname:port --broker-list broker 1, broker 2 --topic-list topic 1, topic 2
```



Important: On secure clusters the admin client config property file has to be specified with the --command-config option. Otherwise, the tool fails to execute.

If no topic is specified with the --topic-list option, then all topics are queried. If no broker is specified with the --broker-list option, then all brokers are queried. If a log directory is offline, the log directory will be marked offline in the script output. Error example:

```
"error": "org.apache.kafka.common.errors.KafkaStorageException"
```

On successful execution, the tool prints a list of partitions per log directory for the specified topics and brokers. The list contains information on topic partition, size, offset lag, and reassignment state. Example output:

```
{
  "brokers": [
    {
      "broker": 86,
      "logDirs": [
        {
          "error": null,
          "logDir": "/var/local/kafka/data",
          "partitions": [
            {
              "isFuture": false,
              "offsetLag": 0,
              "partition": "mytopic1-2",
              "size": 0
            }
          ]
        }
      ]
    }
  ]
}
```

```

    },
    ...
  ],
  "version": 1
}

```

The following table gives an overview of the information provided by the output of the kafka-log-dirs tool.

Table 8: Contents of the kafka-log-dirs Output

Property	Description
broker	Displays the ID of the broker.
error	Indicates if there is a problem with the disk that hosts the topic partition. If an error is detected, org.apache.kafka.common.errors.KafkaStorageException is displayed. If no error is detected, the value is null.
logDir	Specifies the location of the log directory. Returns an absolute path.
isfuture	The reassignment state of the partition. This property shows whether there is currently replica movement underway between the log directories.
offsetLag	Displays the offset lag of the partition.
partition	Displays the name of the partition.
size	Displays the size of the partition in bytes.

zookeeper-security-migration

Learn how to use the zookeeper-security-migration tool.

The zookeeper-security-migration tool is used in the process of restricting or unrestricting access to metadata stored in Zookeeper. When executed, the tool updates the ACLs of znodes based on the configuration specified by the user.



Important: Running the zookeeper-security-migration tool is only one of the steps required when restricting or unrestricting access. For full instructions, see Restricting Access to Kafka Metadata in Zookeeper.

Set the ACLs on all existing Zookeeper znodes to secure with the following command:

```
zookeeper-security-migration --zookeeper.connect hostname:port --zookeeper
.acl secure
```

Set the ACLs on all existing Zookeeper znodes to unsecure with the following command:

```
zookeeper-security-migration --zookeeper.connect hostname:port --zookeeper
.acl unsecure
```

Related Information

[Restricting Access to Kafka Metadata in Zookeeper](#)

kafka-delegation-tokens

Learn how to use the kafka-delegation-tokens tool.

The kafka-delegation-tokens provides the user with the functionality required for using and managing delegation tokens.

The tool can be used to issue, renew, expire, or describe delegation tokens.

Issue, and store for verification

The owner of the token is the currently authenticated principal. A renewer can be specified when requesting the token.

```
kafka-delegation-tokens --bootstrap-server hostname:port --create --max-life-time-period -1 --command-config client.properties --renewer-principal User:user1
```

Renew

Only the owner and the principals that are renewers of the delegation token can extend its validity by renewing it before it expires. A successful renewal extends the Delegation Token's expiration time for another renew-interval, until it reaches its max lifetime. Expired delegation tokens cannot be renewed. The brokers remove expired delegation tokens from the broker's cache and from Zookeeper.



Important: A delegation token cannot be used to renew a delegation token. The delegation token owner must authenticate using a different method to renew a token.

```
kafka-delegation-tokens --bootstrap-server hostname:port --renew --renew-time-period -1 --command-config client.properties --hmac lAYYSFmLs4bTjf+1TZ1LCHR/ZZFNA==
```

Remove

Delegation tokens are removed when they are canceled by the client or when they expire.

```
kafka-delegation-tokens --bootstrap-server hostname:port --expire --expiry-time-period -1 --command-config client.properties --hmac lAYYSFmLs4bTjf+1TZ1LCHR/ZZFNA==
```

Describe

Tokens can be described by owners, renewers or the Kafka super user.

```
kafka-delegation-tokens --bootstrap-server hostname:port --describe --command-config client.properties --owner-principal User:user1
```



Note: In Apache Kafka, principals that have the describe permission on the token resource can also describe the token.

kafka-*-perf-test

The kafka-*-perf-test tools can be used in several ways. In general, it is expected that these tools should be used on a test or development cluster.

The kafka-*-perf-test tools allow you to:

- Measure, read, and write throughput.
- Stress test the cluster based on specific parameters (such as message size).
- Load test for the purpose of evaluating specific metrics or determining the impact of cluster configuration changes.

The kafka-producer-perf-test script can either create a randomly generated byte record:

```
kafka-producer-perf-test --topic TOPIC --record-size SIZE_IN_BYTES
```

or randomly read from a set of provided records:

```
kafka-producer-perf-test --topic TOPIC --payload-delimiter DELIMITER --payload-file INPUT_FILE
```

where the *INPUT_FILE* is a concatenated set of pre-generated messages separated by *DELIMITER*. This script keeps producing messages or limited based on the *--num-records* flag.

The *kafka-consumer-perf-test* is:

```
kafka-consumer-perf-test --broker-list host1:port1,host2:port2,... --zookeeper zk1:port1,zk2:port2,... --topic TOPIC
```

The flags of most interest for this command are:

- *--group gid*: If you run more than one instance of this test, you will want to set different ids for each instance.
- *--num-fetch-threads*: Defaults to 1. Increase if higher throughput testing is needed.
- *--from-latest*: To start consuming from the latest offset. May be needed for certain types of testing.

Configuring log levels for command line tools

Learn how to change the logging level of command line tools.

About this task

In some cases it can prove useful to change the default logging level of the command line tools. This can be done by setting the required logging level in the log4j properties file that the tools use.

Procedure

1. Create a copy of the *log4j.properties* file that the tools use.

You are free to change the location where the copy gets created.

```
cp /etc/kafka/conf/tools-log4j.properties /var/tmp
```

2. Open the copied file and change the value of the *root.logger* property as required.

For example:

```
root.logger=TRACE,console
```

3. Pass the location of the newly created properties file as a JVM parameter.

You can do this by adding *-Dlog4j.configuration=file:[PATH_TO_FILE]* to the *KAFKA_OPTS* variable. For example:

```
export KAFKA_OPTS="-Dlog4j.configuration=file:/var/tmp/tools-log4j.properties"
```

Results

The specified logging level is configured.

Understanding the *kafka-run-class* Bash Script

Almost all the provided Kafka tools eventually call the *kafka-run-class* script. This script is generally not called directly. However, if you are proficient with bash and want to understand certain features available in all Kafka scripts as well as some potential debugging scenarios, familiarity with the *kafka-run-class* script can prove highly beneficial.

For example, there are some useful environment variables that affect all the command line scripts:

- `KAFKA_DEBUG` allows a Java debugger to attach to the JVM launched by the particular script. Setting `KAFKA_DEBUG` also allows some further debugging customization:
 - `JAVA_DEBUG_PORT` sets the JVM debugging port.
 - `JAVA_DEBUG_OPTS` can be used to override the default debugging arguments being passed to the JVM.
- `KAFKA_HEAP_OPTS` can be used to pass memory setting arguments to the JVM.
- `KAFKA_JVM_PERFORMANCE_OPTS` can be used to pass garbage collection flags to the JVM.