cloudera®

Apache Kafka Guide

Important Notice

Cloudera, Inc.
395 Page Mill Road
Palo Alto, CA 94306
info@cloudera.com
US: 1-888-789-1488
Intl: 1-650-362-0488
www.cloudera.com

Release Information

Version: CDH 6.0.x
Date: September 18, 2020

Table of Contents

Apache Kafka Guide

Apache Kafka is a streaming message platform. It is designed to be high performance, highly available, and redundant. Examples of applications that can use such a platform include:

- **Internet of Things.** TVs, refrigerators, washing machines, dryers, thermostats, and personal health monitors can all send telemetry data back to a server through the Internet.
- **Sensor Networks.** Areas (farms, amusement parks, forests) and complex devices (engines) can be designed with an array of sensors to track data or current status.
- **Positional Data.** Delivery trucks or massively multiplayer online games can send location data to a central platform.
- **Other Real-Time Data.** Satellites and medical sensors can send information to a central area for processing.

Ideal Publish-Subscribe System

The ideal publish-subscribe system is straight-forward: Publisher A's messages must make their way to Subscriber A, Publisher B's messages must make their way to Subscriber B, and so on.

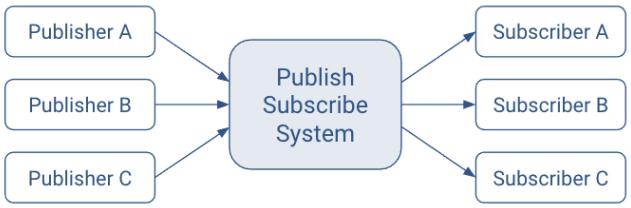


Figure 1: Ideal Publish-Subscribe System

An ideal system has the benefit of:

- **Unlimited Lookback.** A new Subscriber A1 can read Publisher A's stream at any point in time.
- **Message Retention.** No messages are lost.
- **Unlimited Storage.** The publish-subscribe system has unlimited storage of messages.
- **No Downtime.** The publish-subscribe system is never down.
- **Unlimited Scaling.** The publish-subscribe system can handle any number of publishers and/or subscribers with constant message delivery latency.

Now let's see how Kafka's implementation relates to this ideal system.

Kafka Architecture

As is the case with all real-world systems, Kafka's architecture deviates from the ideal publish-subscribe system. Some of the key differences are:

- Messaging is implemented on top of a replicated, distributed commit log.
- The client has more functionality and, therefore, more responsibility.
- Messaging is optimized for batches instead of individual messages.
- Messages are retained even after they are consumed; they can be consumed again.

The results of these design decisions are:

- Extreme horizontal scalability
- Very high throughput
- High availability
- but, different semantics and message delivery guarantees

The next few sections provide an overview of some of the more important parts, while later section describe design specifics and operations in greater detail.

Topics

In the ideal system presented above, messages from one publisher would somehow find their way to each subscriber. Kafka implements the concept of a topic. A topic allows easy matching between publishers and subscribers.

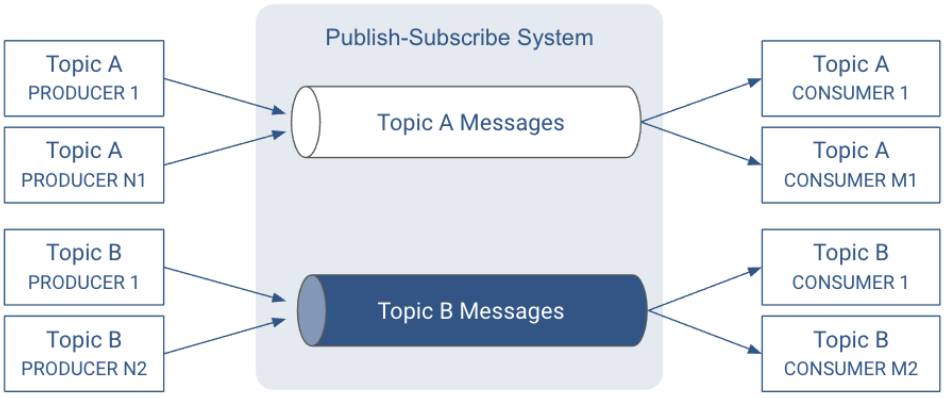


Figure 2: Topics in a Publish-Subscribe System

A topic is a queue of messages written by one or more producers and read by one or more consumers. A topic is identified by its name. This name is part of a global namespace of that Kafka cluster.

Specific to Kafka:

- Publishers are called producers.
- Subscribers are called consumers.

As each producer or consumer connects to the publish-subscribe system, it can read from or write to a specific topic.

Brokers

Kafka is a distributed system that implements the basic features of the ideal publish-subscribe system described above. Each host in the Kafka cluster runs a server called a broker that stores messages sent to the topics and serves consumer requests.

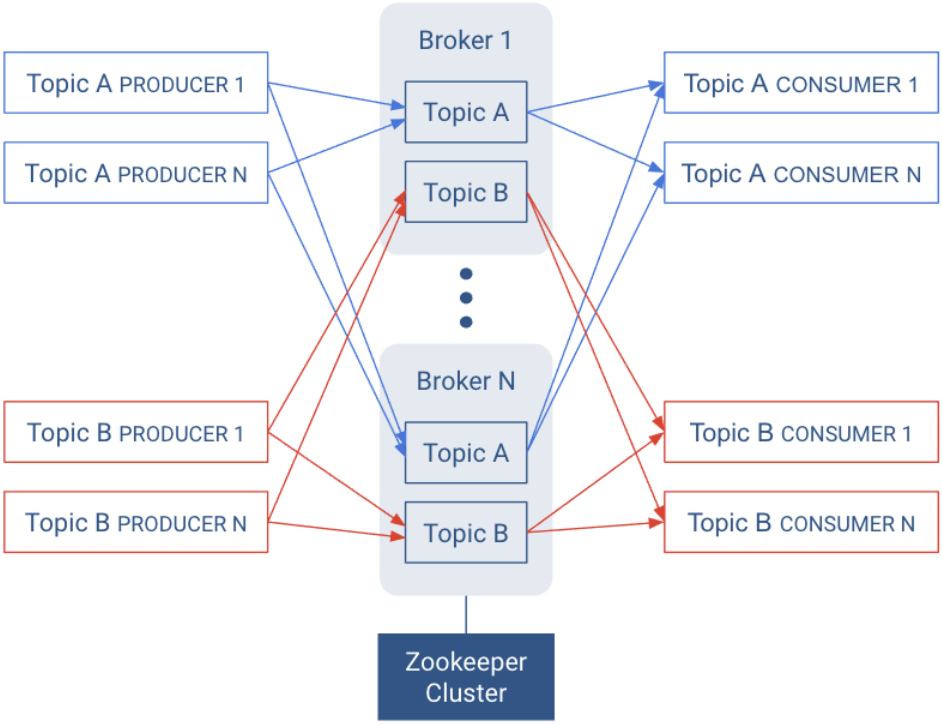


Figure 3: Brokers in a Publish-Subscribe System

Kafka is designed to run on multiple hosts, with one broker per host. If a host goes offline, Kafka does its best to ensure that the other hosts continue running. This solves part of the “No Downtime” and “Unlimited Scaling” goals from the ideal publish-subscribe system.

Kafka brokers all talk to Zookeeper for distributed coordination, additional help for the "Unlimited Scaling" goal from the ideal system.

Topics are replicated across brokers. Replication is an important part of “No Downtime,” “Unlimited Scaling,” and “Message Retention” goals.

There is one broker that is responsible for coordinating the cluster. That broker is called the controller.

As mentioned earlier, an ideal topic behaves as a queue of messages. In reality, having a single queue has scaling issues. Kafka implements partitions for adding robustness to topics.

Records

In Kafka, a publish-subscribe message is called a record. A record consists of a key/value pair and metadata including a timestamp. The key is not required, but can be used to identify messages from the same data source. Kafka stores keys and values as arrays of bytes. It does not otherwise care about the format.

The metadata of each record can include headers. Headers may store application-specific metadata as key-value pairs. In the context of the header, keys are strings and values are byte arrays.

For specific details of the record format, see the Record definition in the Apache Kafka documentation.

Partitions

Instead of all records handled by the system being stored in a single log, Kafka divides records into partitions. Partitions can be thought of as a subset of all the records for a topic. Partitions help with the ideal of “Unlimited Scaling”.

Records in the same partition are stored in order of arrival.

When a topic is created, it is configured with two properties:

partition count

The number of partitions that records for this topic will be spread among.

replication factor

The number of copies of a partition that are maintained to ensure consumers always have access to the queue of records for a given topic.

Each topic has one leader partition. If the replication factor is greater than one, there will be additional follower partitions. (For the replication factor = M, there will be M-1 follower partitions.)

Any Kafka client (a producer or consumer) communicates only with the leader partition for data. All other partitions exist for redundancy and failover. Follower partitions are responsible for copying new records from their leader partitions. Ideally, the follower partitions have an exact copy of the contents of the leader. Such partitions are called in-sync replicas (ISR).

With N brokers and topic replication factor M, then

- If M < N, each broker will have a subset of all the partitions
- If M = N, each broker will have a complete copy of the partitions

In the following illustration, there are N = 2 brokers and M = 2 replication factor. Each producer may generate records that are assigned across multiple partitions.

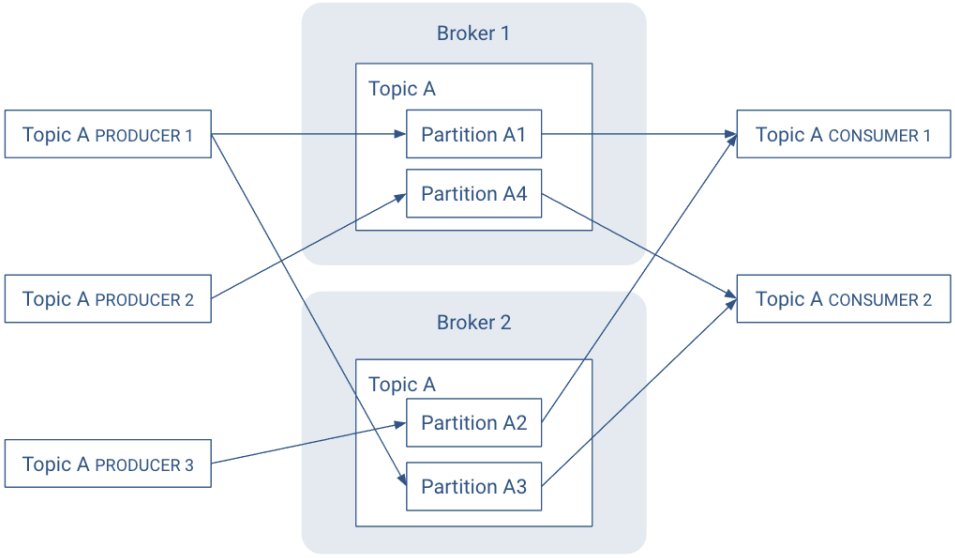


Figure 4: Records in a Topic are Stored in Partitions, Partitions are Replicated across Brokers

Partitions are the key to keeping good record throughput. Choosing the correct number of partitions and partition replications for a topic

- Spreads leader partitions evenly on brokers throughout the cluster
- Makes partitions within the same topic are roughly the same size.
- Balances the load on brokers.

Record Order and Assignment

By default, Kafka assigns records to a partitions round-robin. There is no guarantee that records sent to multiple partitions will retain the order in which they were produced. Within a single consumer, your program will only have record ordering within the records belonging to the same partition. This tends to be sufficient for many use cases, but does add some complexity to the stream processing logic.

Tip: Kafka guarantees that records in the same partition will be in the same order in all replicas of that partition.

If the order of records is important, the producer can ensure that records are sent to the same partition. The producer can include metadata in the record to override the default assignment in one of two ways:

- The record can indicate a specific partition.
- The record can includes an assignment key.

  The hash of the key and the number of partitions in the topic determines which partition the record is assigned to. Including the same key in multiple records ensures all the records are appended to the same partition.

Logs and Log Segments

Within each topic, each partition in Kafka stores records in a log structured format. Conceptually, each record is stored sequentially in this type of "log".

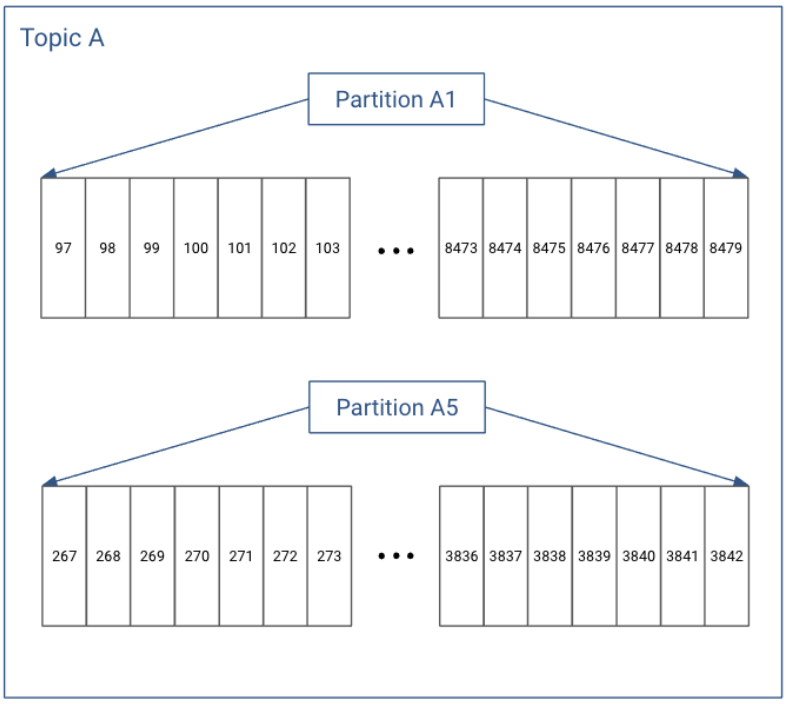


Figure 5: Partitions in Log Structured Format

> **Note:** These references to "log" should not be confused with where the Kafka broker stores their operational logs.

In actuality, each partition does not keep all the records sequentially in a single file. Instead, it breaks each log into log segments. Log segments can be defined using a size limit (for example, 1 GB), as a time limit (for example, 1 day), or both. Administration around Kafka records often occurs at the log segment level.

Each of the partitions is broken into segments, with Segment N containing the most recent records and Segment 1 containing the oldest retained records. This is configurable on a per-topic basis.

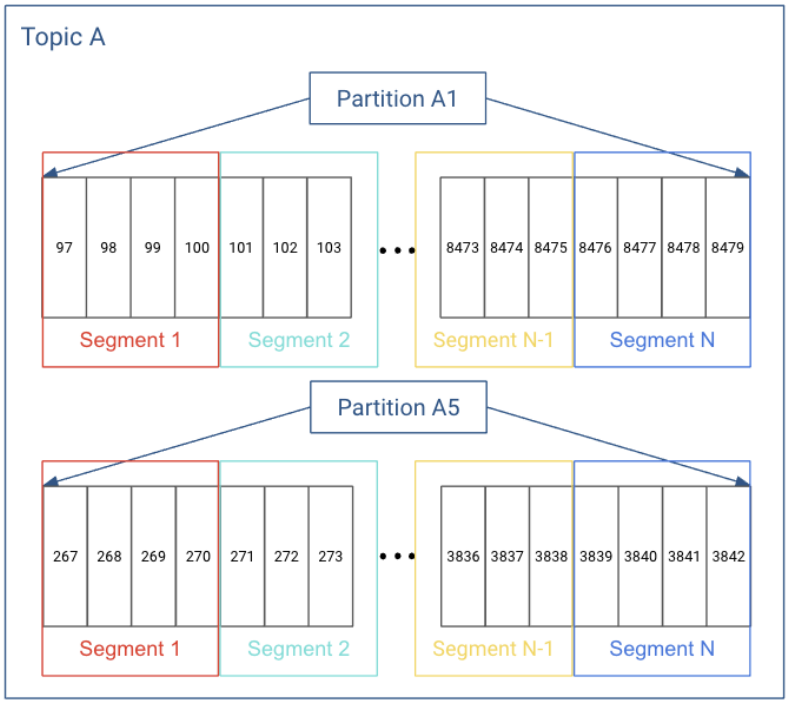


Figure 6: Partition Log Segments

Kafka Brokers and ZooKeeper

The broker, topic, and partition information are maintained in Zookeeper. In particular, the partition information, including partition and replica locations, updates fairly frequently. Because of frequent metadata refreshes, the connection between the brokers and the Zookeeper cluster needs to be reliable. Similarly, if the Zookeeper cluster has other intensive processes running on it, that can add sufficient latency to the broker/Zookeeper interactions to cause issues.

- Kafka Controller maintains leadership via Zookeeper (shown in orange)
- Kafka Brokers also store other relevant metadata in Zookeeper (also in orange)
- Kafka Partitions maintain replica information in Zookeeper (shown in blue)

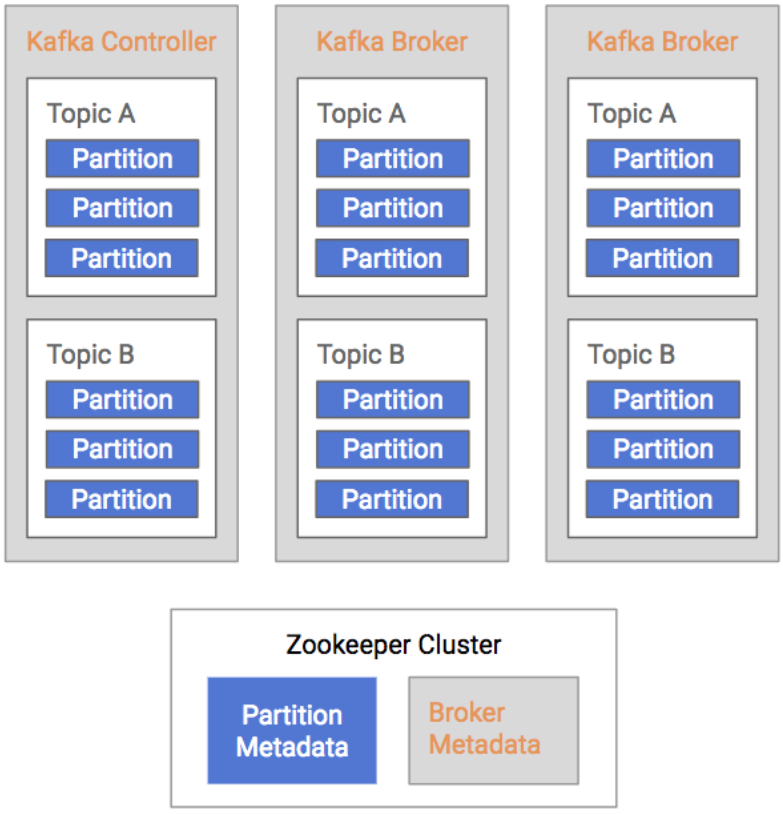


Figure 7: Broker/ZooKeeper Dependencies

Kafka Setup

Hardware Requirements

Kafka can function on a fairly small amount of resources, especially with some configuration tuning. Out of the box configurations can run on little as 1 core and 1 GB memory with storage scaled based on data retention requirements. These are the defaults for both broker and Mirror Maker in Cloudera Manager version 6.x.

Brokers

Kafka requires a fairly small amount of resources, especially with some configuration tuning. By default, Kafka, can run on as little as 1 core and 1GB memory with storage scaled based on requirements for data retention.

CPU is rarely a bottleneck because Kafka is I/O heavy, but a moderately-sized CPU with enough threads is still important to handle concurrent connections and background tasks.

Kafka brokers tend to have a similar hardware profile to HDFS data nodes. How you build them depends on what is important for your Kafka use cases. Use the following guidelines:

To affect performance of these features:	Adjust these parameters:
Message Retention	Disk size
Client Throughput (Producer & Consumer)	Network capacity
Producer throughput	Disk I/O
Consumer throughput	Memory

A common choice for a Kafka node is as follows:

Component	Memory/Java Heap	CPU	Disk
Broker	• RAM: 64 GB • Recommended Java heap: 4 GB Set this value using the **Java Heap Size of Broker** Kafka configuration property. See Other Kafka Broker Properties table.	12- 24 cores	• 1 HDD For operating system • 1 HDD for Zookeeper dataLogDir • 10- HDDs, using Raid 10, for Kafka data
MirrorMaker	1 GB heap Set this value using the **Java Heap Size of MirrorMaker** Kafka configuration property.	1 core per 3-4 streams	No disk space needed on MirrorMaker instance. Destination brokers should have sufficient disk space to store the topics being copied over.

Networking requirements: Gigabit Ethernet or 10 Gigabit Ethernet. Avoid clusters that span multiple data centers.

ZooKeeper

It is common to run ZooKeeper on 3 broker nodes that are dedicated for Kafka. However, for optimal performance Cloudera recommends the usage of dedicated Zookeeper hosts. This is especially true for larger, production environments.

Kafka Performance Considerations

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:

Do not run 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 highly recommended to not use VMs for Kafka; if you are running Kafka in a virtual environment you will need to rely on your VM vendor for help 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

Operating System Requirements

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 these in Cloudera Manager via **Kafka** > **Configuration** > **Maximum Process File Descriptors**. We recommend a configuration of 100000 or higher.

- Max Memory Map

  You must configure this in your specific kernel settings. We recommend a configuration of 32000 or higher.

- Max Socket Buffer Size

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

Kafka in Cloudera Manager

Cloudera Manager is the recommended way to administer your cluster, including administering Kafka. When you open the Kafka service from Cloudera Manager, you see the following dashboard:

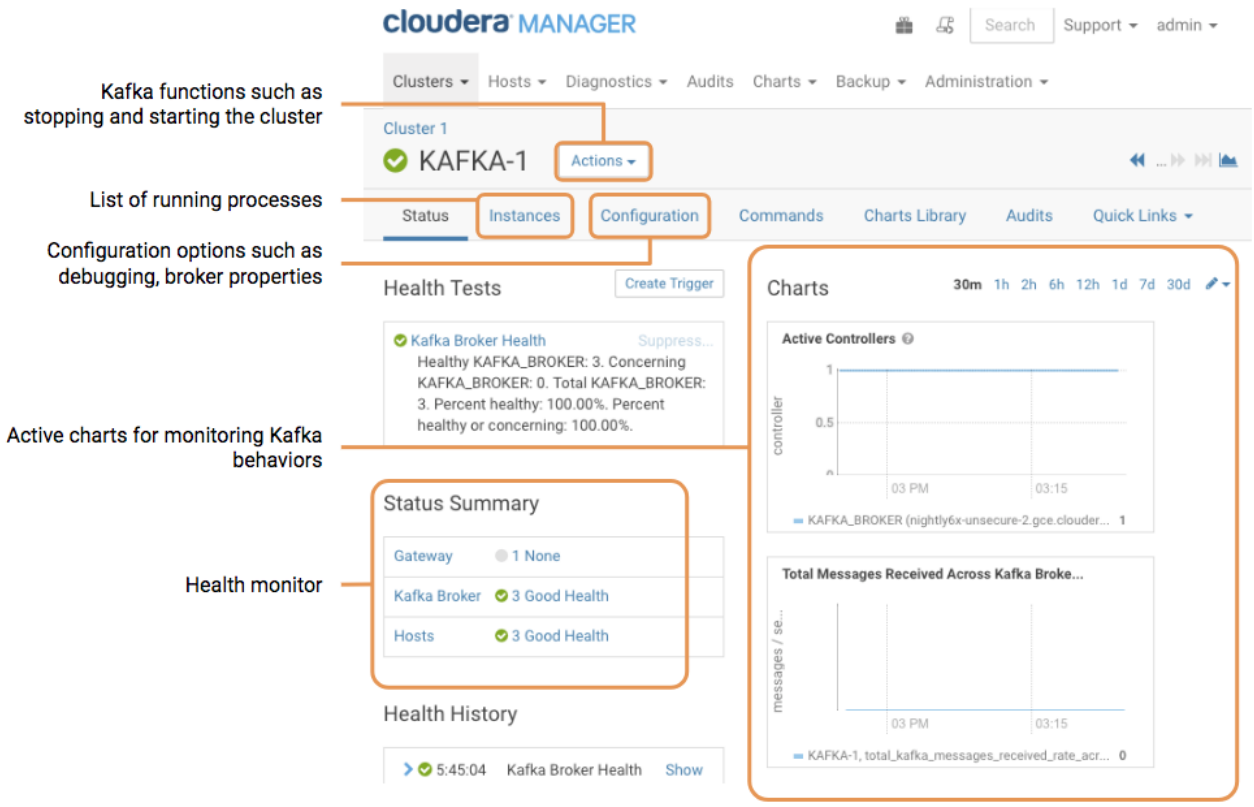


Figure 8: Kafka Service in Cloudera Manager

Kafka Clients

Kafka clients are created to read data from and write data to the Kafka system. Clients can be producers, which publish content to Kafka topics. Clients can be subscribers, which read content from Kafka topics.

Commands for Client Interactions

Assuming you have Kafka running on your cluster, here are some commands that describe the typical steps you would need to exercise Kafka functionality:

Create a topic

```
kafka-topics --create --zookeeper zkinfo --replication-factor 1 --partitions 1 --topic
 test
```

where the ZooKeeper connect string *zkinfo* is a comma-separated list of the Zookeeper nodes in *host*: *port* format.

Validate the topic was created successfully

```
kafka-topics --list --zookeeper zkinfo
```

Produce messages

The following command can be used to publish a message to the Kafka cluster. After the command, each typed line is a message that is sent to Kafka. After the last message, send an EOF or stop the command with Ctrl-D.

```
$ kafka-console-producer --broker-list kafkainfo --topic test
My first message.
My second message.
^D
```

where *kafkainfo* is a comma-separated list of the Kafka brokers in *host*:*port* format. Using more than one makes sure that the command can find a running broker.

Consume messages

The following command can be used to subscribe to a message from the Kafka cluster.

```
kafka-console-consumer --bootstrap-server kafkainfo --topic test --from-beginning
```

The output shows the same messages that you entered during your producer.

Set a ZooKeeper root node

It's possible to use a root node (`chroot`) for all Kafka nodes in ZooKeeper by setting a value for `zookeeper.chroot` in Cloudera Manager. Append this value to the end of your ZooKeeper connect string.

Set `chroot` in Cloudera Manager:

```
zookeeper.chroot=/kafka
```

Form the ZooKeeper connect string as follows:

```
--zookeeper zkinfo/kafka
```

If you set `chroot` and then use only the host and port in the connect string, you'll see the following exception:

```
InvalidReplicationFactorException: replication factor: 3 larger than available brokers:
 0
```

Kafka Producers

Kafka producers are the publishers responsible for writing records to topics. Typically, this means writing a program using the KafkaProducer API. To instantiate a producer:

```
KafkaProducer<String, String> producer = new
KafkaProducer<>(producerConfig);
```

Most of the important producer settings, and mentioned below, are in the configuration passed by this constructor.

Serialization of Keys and Values

For each producer, there are two serialization properties that must be set, `key.serializer` (for the key) and `value.serializer` (for the value). You can write custom code for serialization or use one of the ones already provided by Kafka. Some of the more commonly used ones are:

- `ByteArraySerializer`: Binary data
- `StringSerializer`: String representations

Managing Record Throughput

There are several settings to control how many records a producer accumulates before actually sending the data to the cluster. This tuning is highly dependent on the data source. Some possibilities include:

- `batch.size`: Combine this fixed number of records before sending data to the cluster.
- `linger.ms`: Always wait at least this amount of time before sending data to the cluster; then send however many records has accumulated in that time.
- `max.request.size`: Put an absolute limit on data size sent. This technique prevents network congestion caused by a single transfer request containing a large amount of data relative to the network speed.
- `compression.type`: Enable compression of data being sent.
- `retries`: Enable the client for retries based on transient network errors. Used for reliability.

Acknowledgments

The full write path for records from a producer is to the leader partition and then to all of the follower replicas. The producer can control which point in the path triggers an acknowledgment. Depending on the `acks` setting, the producer may wait for the write to propagate all the way through the system or only wait for the earliest success point.

Valid `acks` values are:

- `0`: Do not wait for any acknowledgment from the partition (fastest throughput).
- `1`: Wait only for the leader partition response.
- `all`: Wait for follower partitions responses to meet minimum (slowest throughput).

Partitioning

In Kafka, the partitioner determines how records map to partitions. Use the mapping to ensure the order of records within a partition and manage the balance of messages across partitions. The default partitioner uses the entire key to determine which partition a message corresponds to. Records with the same key are always mapped to the same partition (assuming the number of partitions does not change for a topic). Consider writing a custom partitioner if you have information about how your records are distributed that can produce more efficient load balancing across partitions. A custom partitioner lets you take advantage of the other data in the record to control partitioning.

If a partitioner is not provided to the KafkaProducer, Kafka uses a default partitioner.

The ProducerRecord class is the actual object processed by the KafkaProducer. It takes the following parameters:

- `Kafka Record`: The key and value to be stored.
- `Intended Destination`: The destination topic and the specific partition (optional).

Kafka Consumers

Kafka consumers are the subscribers responsible for reading records from one or more topics and one or more partitions of a topic. Consumers subscribing to a topic can happen manually or automatically; typically, this means writing a program using the KafkaConsumer API.

To instantiate a consumer:

```
KafkaConsumer<String, String> kafkaConsumer = new
KafkaConsumer<>(consumerConfig);
```

The KafkaConsumer class has two generic type parameters. Just as producers can send data (the values) with keys, the consumer can read data by keys. In this example both the keys and values are strings. If you define different types, you need to define a deserializer to accommodate the alternate types. For deserializers you need to implement the `org.apache.kafka.common.serialization.Deserializer` interface.

The most important configuration parameters that we need to specify are:

- `bootstrap.servers`: A list of brokers to initially connect to. List 2 to 3 brokers; you don't needed to list the full cluster.
- `group.id`: Every consumer belongs to a group. That way they'll share the partitions of a topic.
- `key.deserializer/value.deserializer`: Specify how to convert the Java representation to a sequence of bytes to send data through the Kafka protocol.

Subscribing to a topic

Subscribing to a topic using the `subscribe()` method call:

```
kafkaConsumer.subscribe(Collections.singletonList(topic), rebalanceListener);
```

Here we specify a list of topics that we want to consume from and a 'rebalance listener.' Rebalancing is an important part of the consumer's life. Whenever the cluster or the consumers' state changes, a rebalance will be issued. This will ensure that all the partitions are assigned to a consumer.

After subscribing to a topic, the consumer polls to see if there are new records:

```
while (true) {
  data = kafkaConsumer.poll();
  // do something with 'data'
}
```

The poll returns multiple records that can be processed by the client. After processing the records the client commits offsets synchronously, thus waiting until processing completes before continuing to poll.

The last important point is to save the progress. This can be done by the `commitSync()` and `commitAsync()` methods respectively.

```
PLACEHOLDER FOR CODE SNIPPET
```

Auto commit is not recommended; manual commit is appropriate in the majority of use cases.

Groups and Fetching

Kafka consumers are usually assigned to a group. This happens statically by setting the `group.id` configuration property in the consumer configuration. Consuming with groups will result in the consumers balancing the load in the group. That means each consumer will have their fair share of partitions. Also it can never be more consumers than partitions as that way there would be idling consumers.

As shown in the figure below, both consumer groups share the partitions and each partition multicasts messages to both consumer groups. The consumers pull messages from the broker instead of the broker periodically pushing what is available. This helps the consumer as it won't be overloaded and it can query the broker at its own speed. Furthermore, to avoid tight looping, it uses a so called "long-poll". The consumer sends a fetch request to poll for data and receives a reply only when enough data accumulates on the broker.

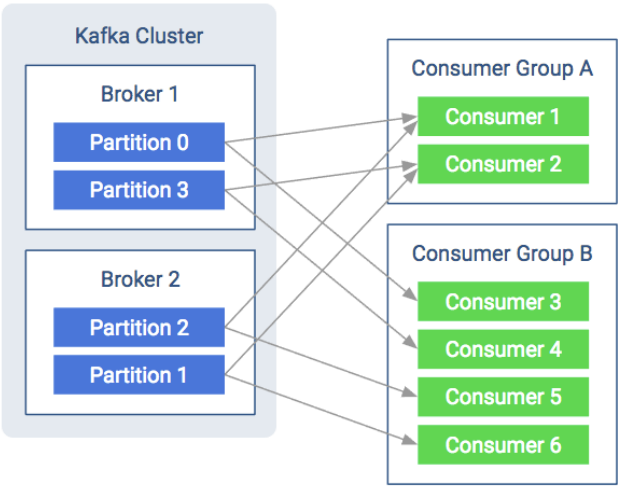


Figure 9: Consumer Groups and Fetching from Partitions

Protocol between Consumer and Broker

This section details how the protocol works, what messages are going on the wire and how that contributes to the overall behavior of the consumer. When discussing the internals of the consumers, there are a couple of basic terms to know:

Heartbeat

When the consumer is alive and is part of the consumer group, it sends heartbeats. These are short periodic messages that tell the brokers that the consumer is alive and everything is fine.

Session

Often one missing heartbeat is not a big deal, but how do you know if a consumer is not sending heartbeats for long enough to indicate a problem? A session is such a time interval. If the consumer didn't send any heartbeats for longer than the session, the broker can consider the consumer dead and remove it from the group.

Coordinator

The special broker which manages the group on the broker side is called the coordinator. The coordinator handles heartbeats and assigns the leader. Every group has a coordinator that organizes the startup of a consumer group and assist whenever a consumer leaves the group.

Leader

The leader consumer is elected by the coordinator. Its job is to assign partitions to every consumer in the group at startup or whenever a consumer leaves or joins the group. The leader holds the assignment strategy, it is decoupled from the broker. That means consumers can reconfigure the partition assignment strategy without restarting the brokers.

Startup Protocol

As mentioned before, the consumers are working usually in groups. So a major part of the startup process is spent with figuring out the consumer group.

At startup, the first step is to match protocol versions. It is possible that the broker and the consumer are of different versions (the broker is older and the consumer is newer, or vice versa). This matching is done by the `API_VERSIONS` request.

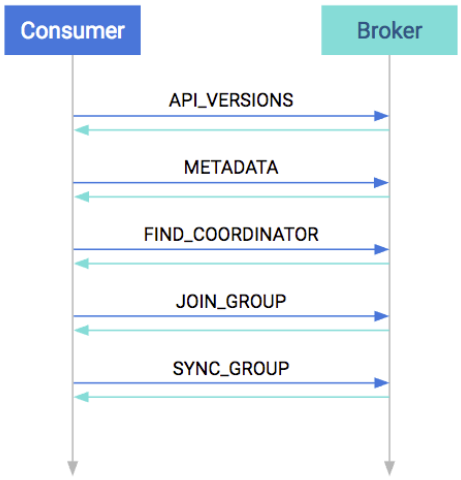


Figure 10: Startup Protocol

The next step is to collect cluster information, such as the addresses of all the brokers (prior to this point we used the bootstrap server as a reference), partition counts, and partition leaders. This is done in the `METADATA` request.

After acquiring the metadata, the consumer has the information needed to join the group. By this time on the broker side, a coordinator has been selected per consumer group. The consumers must find their coordinator with the `FIND_COORDINATOR` request.

After finding the coordinator, the consumer(s) are ready to join the group. Every consumer in the group sends their own member-specific metadata to the coordinator in the `JOIN_GROUP` request. The coordinator waits until all the consumers have sent their request, then assigns a leader for the group. At the response plus the collected metadata are sent to the leader, so it knows about its group.

The remaining step is to assign partitions to consumers and propagate this state. Similar to the previous request, all consumers send a `SYNC_GROUP` request to the coordinator; the leader provides the assignments in this request. After it receives the sync request from each group member, the coordinator propagates this member state in the response. By the end of this step, the consumers are ready and can start consuming.

Consumption Protocol

When consuming, the first step is to query where should the consumer start. This is done in the `OFFSET_FETCH` request. This is not mandatory: the consumer can also provide the offset manually. After this, the consumer is free to pull data from the broker. Data consumption happens in the `FETCH` requests. These are the long-pull requests. They are answered only when the broker has enough data; the request can be outstanding for a longer period of time.

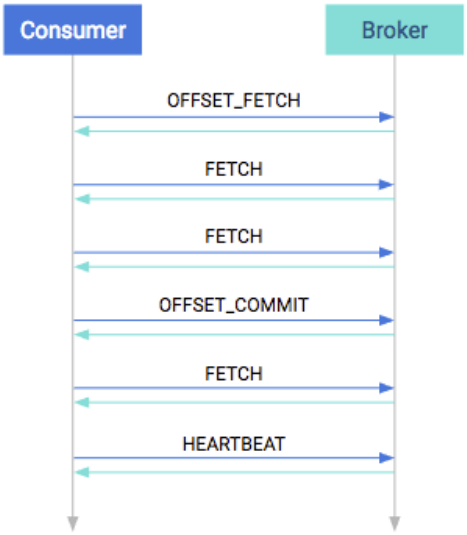


Figure 11: Consumption Protocol

From time to time, the application has to either manually or automatically save the offsets in an `OFFSET_COMMIT` request and send heartbeats too in the `HEARTBEAT` requests. The first ensures that the position is saved while the latter ensures that the coordinator knows that the consumer is alive.

Shutdown Protocol

The last step when the consumption is done is to shut down the consumer gracefully. This is done in one single step, called the `LEAVE_GROUP` protocol.

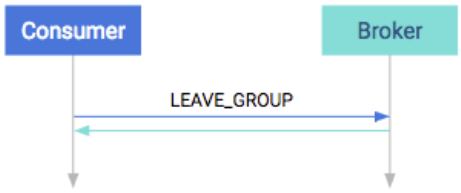


Figure 12: Shutdown Protocol

Rebalancing Partitions

You may notice that there are multiple points in the protocol between consumers and brokers where failures can occur. There are points in the normal operation of the system where you need to change the consumer group assignments. For example, to consume a new partition or to respond to a consumer going offline. The process or responding to cluster information changing is called rebalance. It can occur in the following cases:

- **A consumer leaves.** It can be a software failure where the session times out or a connection stalls for too long, but it can also be a graceful shutdown.
- **A consumer joins.** It can be a new consumer but an old one that just recovered from a software failure (automatically or manually).
- **Partition is adjusted.** A partition can simply go offline because of a broker failure or a partition coming back online. Alternatively an administrator can add or remove partitions to/from the broker. In these cases the consumers must reassign who is consuming.
- **The cluster is adjusted.** When a broker goes offline, the partitions that are lead by this broker will be reassigned. In turn the consumers must rebalance so that they consume from the new leader. When a broker comes back, then eventually a preferred leader election happens which restores the original leadership. The consumers must follow this change as well.

On the consumer side, this rebalance is propagated to the client via the `ConsumerRebalanceListener` interface. It has two methods. The first, `onPartitionsRevoked`, will be invoked when any partition goes offline. This call happens before the changes would reflect in any of the consumers, so this is the chance to save offsets if manual offset commit is used. On the other hand `onPartitionsAssigned` is invoked after partition reassignment. This would allow for the programmer to detect which partitions are currently assigned to the current consumer. Complete examples can be found in the development section.

Consumer Configuration Properties

There are some very important configurations that any user of Kafka must know:

- `heartbeat.interval.ms`: The interval of the heartbeats. For example, if the heartbeat interval is set to 3 seconds, the consumer sends a short heartbeat message to the broker every 3 seconds to indicate that it is alive.
- `session.timeout.ms`: The consumer tells this timeout to the coordinator. This is used to control the heartbeats and remove the dead consumers. If it's set to 10 seconds, the consumer can miss sending 2 heartbeats, assuming the previous heartbeat setting. If we increase the timeout, the consumer has more room for delays but the broker notices lagging consumers later.
- `max.poll.interval.ms`: It is a very important detail: the consumers must maintain polling and should never do long-running processing. If a consumer is taking too much time between two polls, it will be detached from the consumer group. We can tune this configuration according to our needs. Note that if a consumer is stuck in processing, it will be noticed later if the value is increased.
- `request.timeout.ms`: Generally every request has a timeout. This is an upper bound that the client waits for the server's response. If this timeout elapses, then retries might happen if the number of retries are not exhausted.

Retries

In Kafka retries typically happen on only for certain kinds of errors. When a retriable error is returned, the clients are constrained by two facts: the timeout period and the backoff period.

The timeout period tells how long the consumer can retry the operation. The backoff period how often the consumer should retry. There is no generic approach for "number of retries." Number of retries are usually controlled by timeout periods.

Kafka Clients and ZooKeeper

The default consumer model provides the metadata for offsets in the Kafka cluster. There is a topic named `__consumer_offsets` that the Kafka Consumers write their offsets to.

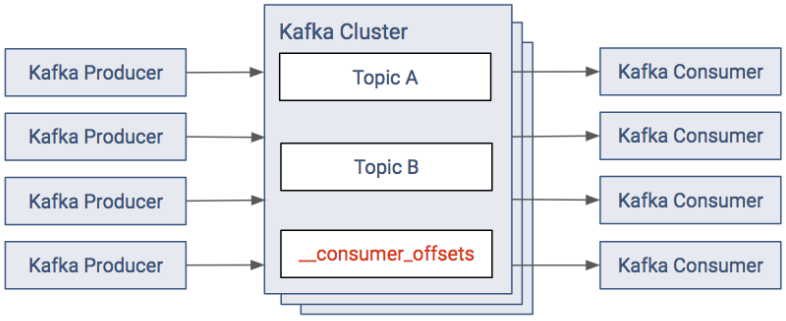


Figure 13: Kafka Consumer Dependencies

In releases before version 2.0 of CDK Powered by Apache Kafka, the same metadata was located in ZooKeeper. The new model removes the dependency and load from Zookeeper. In the old approach:

- The consumers save their offsets in a "consumer metadata" section of ZooKeeper.
- With most Kafka setups, there are often a large number of Kafka consumers. The resulting client load on ZooKeeper can be significant, therefore this solution is discouraged.

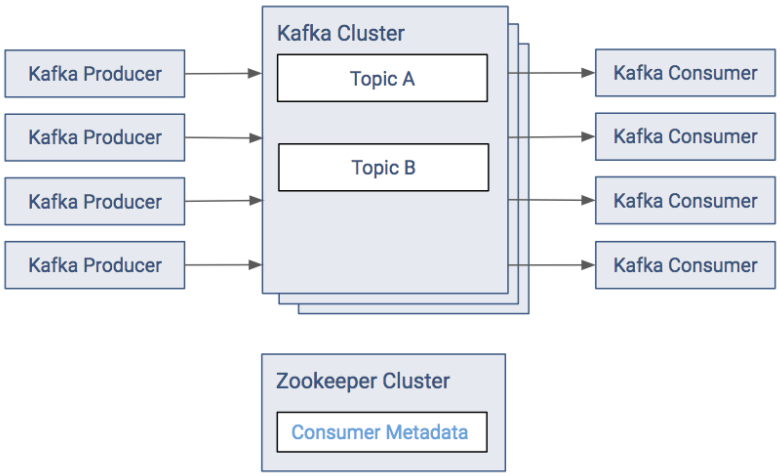


Figure 14: Kafka Consumer Dependencies (Old Approach)

Kafka Brokers

This section covers some of how a broker operates in greater detail. As we go over some of these details, we will illustrate how these pieces can cause brokers to have issues.

Single Cluster Scenarios

The figure below shows a simplified version of a Kafka cluster in steady state. There are N brokers, two topics with nine partitions each. The M replicated partitions are not shown for simplicity. This is going to be the baseline for discussions in later sections.

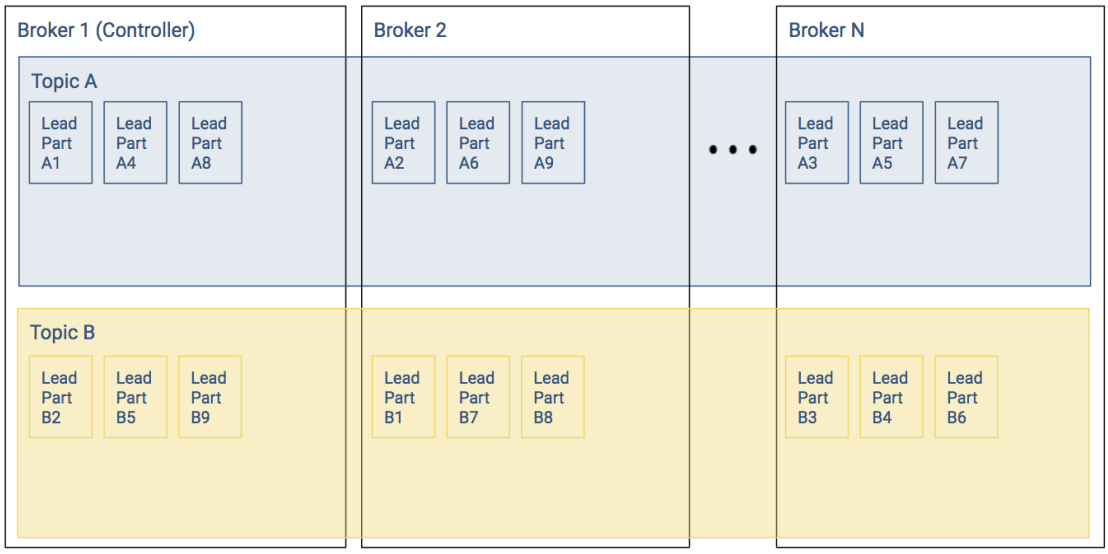


Figure 15: Kafka Cluster in Steady State

Leader Positions

In the baseline example, each broker shown has three partitions per topic. In the figure above, the Kafka cluster has well balanced leader partitions. Recall the following:

- Producer writes and consumer reads occur at the partition level
- Leader partitions are responsible for ensuring that the follower partitions keep their records in sync

In the baseline example, since the leader partitions were evenly distributed, most of the time the load to the overall Kafka cluster will be relatively balanced.

In the example below, since a large chunk of the leaders for Topic A and Topic B are on Broker 1, a lot more of the overall Kafka workload will occur at Broker 1. This will cause a backlog of work, which slows down the cluster throughput, which will worsen the backlog.

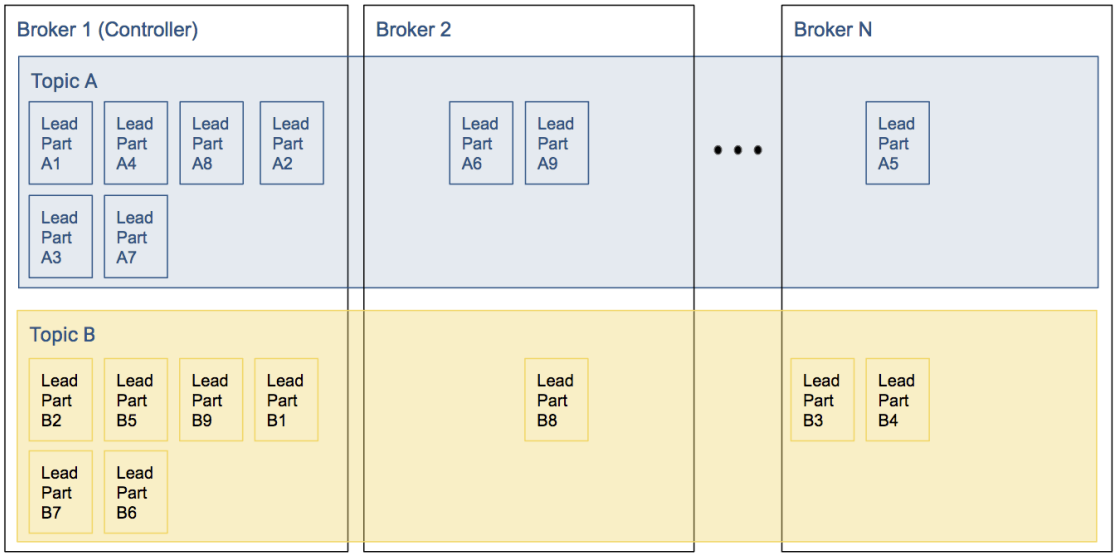


Figure 16: Kafka Cluster with Leader Partition Imbalance

Even if a cluster starts with perfectly balanced topics, failures of brokers can cause these imbalances: if leader of a partition goes down one of the replicas will become the leader. When the original (preferred) leader comes back, it will get back leadership only if automatic leader rebalancing is enabled; otherwise the node will become a replica and the cluster gets imbalanced.

In-Sync Replicas

Let's look at Topic A from the previous example with follower partitions:

- Broker 1 has six leader partitions, broker 2 has two leader partitions, and broker 3 has one leader partition.
- Assuming a replication factor of 3.

Assuming all replicas are in-sync, then any leader partition can be moved from Broker 1 to another broker without issue. However, in the case where some of the follower partitions have not caught up, then the ability to change leaders or have a leader election will be hampered.

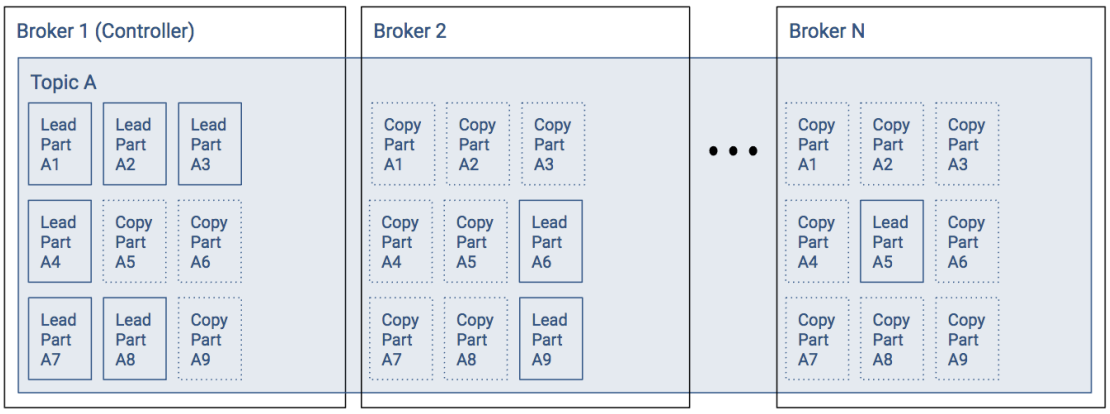


Figure 17: Kafka Topic with Leader and Follower Partitions

Topic Configuration

We already introduced the concept of topics. When managing a Kafka cluster, configuring a topic can require some planning. For small clusters or low record throughput, topic planning isn't particularly tricky, but as you scale to the large clusters and high record throughput, such planning becomes critical.

Topic Creation

To be able to use a topic, it has to be created. This can happen automatically or manually. When enabled, the Kafka cluster creates topics on demand.

Automatic Topic Creation

If `auto.create.topics.enable` is set to `true` and a client is requesting metadata about a non-existent topic, then the broker will create a topic with the given name. In this case, its replication factor and partition count is derived from the broker configuration. Corresponding configuration entries are `default.replication.factor` and `num.partitions`. The default value for each these properties is 1. This means the topic will not scale well and will not be tolerant to broker failure. It is recommended to set these default value higher or even better switching off this feature and creating topics manually with configuration suited for the use case at hand.

Manual topic creation

You can create topics from the command line with `kafka-topics` tool. Specify a replication factor and partition count, with optional configuration overrides. Alternatively, for each partition, you can specify which brokers have to hold a copy of that partition.

Topic Properties

There are numerous properties that influence how topics are handled by the cluster. These can be set with `kafka-topics` tool on topic creation or later on with `kafka-configs`. The most commonly used properties are:

- `min.insync.replicas`: specifies how many brokers have to replicate the records before the leader sends back an acknowledgment to the producer (if producer property `acks` is set to `all`). With a replication factor of 3, a minimum in-sync replicas of 2 guarantees a higher level of durability. It is not recommended that you set this value equal to the replication factor as it makes producing to the topic impossible if one of the brokers is temporarily down.
- `retention.bytes` and `retention.ms`: determines when a record is considered outdated. When data stored in one partition exceeds given limits, broker starts a cleanup to save disk space.
- `segment.bytes` and `segment.ms`: determines how much data is stored in the same log segment (that is, in the same file). If any of these limits is reached, a new log segment is created.
- `unclean.leader.election.enable`: if `true`, replicas that are not in-sync may be elected as new leaders. This only happens when there are no live replicas in-sync. As enabling this feature may result in data loss, it should be switched on only if availability is more important than durability.
- `cleanup.policy`: either `delete` or `compact`. If `delete` is set, old log segments will be deleted. Otherwise, only the latest record is retained. This process is called *log compaction*. This is covered in greater detail in the Record Management on page 60 section.

If you do not specify these properties, the prevailing broker level configuration will take effect. A complete list of properties can be found in the Topic-Level Configs section of the Apache Kafka documentation.

Partition Management

Partitions are at the heart of how Kafka scales performance. Some of the administrative issues around partitions can be some of the biggest challenges in sustaining high performance.

When creating a topic, you specify which brokers should have a copy of which partition or you specify replication factor and number of partitions and the controller generates a replica assignment for you. If there are multiple brokers that are assigned a partition, the first one in the list is always the preferred leader.

Whenever the leader of a partition goes down, Kafka moves leadership to another broker. Whether this is possible depends on the current set of in-sync replicas and the value of `unclean.leader.election.enable`. However, no new Kafka broker will start to replicate the partition to reach replication factor again. This is to avoid unnecessary load on brokers when one of them is temporarily down. Kafka will regularly try to balance leadership between brokers by electing the preferred leader. But this balance is based on number of leaderships and not throughput.

Partition Reassignment

In some cases require manual reassignment of partitions:

- If the initial distribution of partitions and leaderships creates an uneven load on brokers.
- If you want to add or remove brokers from the cluster.

Use `kafka-reassign-partitions` tool to move partitions between brokers. The typical workflow consist of the following:

- Generate a reassignment file by specifying topics to move and which brokers to move to (by setting `--topic-to-move-json-file` and `--broker-list` to `--generate` command).
- Optionally edit the reassignment file and verify it with the tool.
- Actually re-assigning partitions (with option `--execute`).
- Verify if the process has finished as intended (with option `--verify`).

Note: When specifying throttles for inter broker communication, make sure you use the command with `--verify` option to remove limitations on replication speed.

Adding Partitions

You can use `kafka-topics` tool to increase the number of partitions in a given topic. However, note that **adding partitions will in most cases break the guarantee preserving the order of records with the same key**, because it changes which partition a record key is produced to. Although order of records is preserved for both the old partition the key was produced to and the new one, it still might happen that records from the new partition are consumed before records from the old one.

Choosing the Number of Partitions

When choosing the number of partitions for a topic, you have to consider the following:

- More partitions mean higher throughput.
- You should not have more than a few tens of thousands of partitions in a Kafka cluster.
- In case of an unclean shutdown of one of the brokers, the partitions it was leader for have to be led by other brokers and moving leadership for a few thousand partitions one by one can take seconds of unavailability.
- Kafka keeps all log segment files open at all times. More partitions can mean more file handles to have open.
- More partitions can cause higher latency.

Controller

The controller is one of the brokers that has additional partition and replica management responsibilities. It will control / be involved whenever partition metadata or state is changed, such as when:

- Topics or partitions are created or deleted.
- Brokers join or leave the cluster and partition leader or replica reassignment is needed.

It also tracks the list of in sync replicas (ISRs) and maintains broker, partition, and ISR data in Zookeeper.

Controller Election

Any of the brokers can play the role of the controller, but in a healthy cluster there is exactly one controller. Normally this is the broker that started first, but there are certain situations when a re-election is needed:

- If the controller is shut down or crashes.
- If it loses connection to Zookeeper.

When a broker starts or participates in controller reelection, it will attempt to create an ephemeral node (“`/controller`”) in ZooKeeper. If it succeeds, the broker becomes the controller. If it fails, there is already a controller, but the broker will watch the node.

If the controller loses connection to ZooKeeper or stops ZooKeeper will remove the ephemeral node and the brokers will get a notification to start a controller election.

Every controller election will increase the "*controller epoch*". The controller epoch is used to detect situations when there are multiple active controllers: if a broker gets a message with a lower epoch than the result of the last election, it can be safely ignored. It is also used to detect a "split brain" situation when multiple nodes believe that they are in the controller role.

Having 0 or 2+ controllers means the cluster is in a critical state, as broker and partition state changes are blocked. Therefore it's important to ensure that the controller has a stable connection to ZooKeeper to avoid controller elections as much as possible.

Kafka Integration

Kafka Security

Client-Broker Security with TLS

Kafka allows clients to connect over TLS. By default, TLS is disabled, but can be turned on as needed.

Step 1: Generating Keys and Certificates for Kafka Brokers

Generate the key and the certificate for each machine in the cluster using the Java keytool utility. See Generate TLS Certificates.

Make sure that the common name (CN) matches the fully qualified domain name (FQDN) of your server. The client compares the CN with the DNS domain name to ensure that it is connecting to the correct server.

Step 2: Creating Your Own Certificate Authority

You have generated a public-private key pair for each machine and a certificate to identify the machine. However, the certificate is unsigned, so an attacker can create a certificate and pretend to be any machine. Sign certificates for each machine in the cluster to prevent unauthorized access.

A Certificate Authority (CA) is responsible for signing certificates. A CA is similar to a government that issues passports. A government stamps (signs) each passport so that the passport becomes difficult to forge. Similarly, the CA signs the certificates, and the cryptography guarantees that a signed certificate is computationally difficult to forge. If the CA is a genuine and trusted authority, the clients have high assurance that they are connecting to the authentic machines.

```
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
```

The generated CA is a public-private key pair and certificate used to sign other certificates.

Add the generated CA to the client truststores so that clients can trust this CA:

```
keytool -keystore {client.truststore.jks} -alias CARoot -import -file {ca-cert}
```

Note: If you configure Kafka brokers to require client authentication by setting `ssl.client.auth` to be requested or required on the Kafka brokers config, you must provide a truststore for the Kafka brokers as well. The truststore must have all the CA certificates by which the clients keys are signed. The keystore created in step 1 stores each machine's own identity. In contrast, the truststore of a client stores all the certificates that the client should trust. Importing a certificate into a truststore means trusting all certificates that are signed by that certificate. This attribute is called the chain of trust. It is particularly useful when deploying SSL on a large Kafka cluster. You can sign all certificates in the cluster with a single CA, and have all machines share the same truststore that trusts the CA. That way, all machines can authenticate all other machines.

Step 3: Signing the Certificate

Now you can sign all certificates generated by step 1 with the CA generated in step 2.

1. Create a certificate request from the keystore:

```
keytool -keystore server.keystore.jks -alias localhost -certreq -file cert-file
```

where:

- *keystore*: the location of the keystore

- *cert-file*: the exported, unsigned certificate of the server

2. Sign the resulting certificate with the CA (in the real world, this can be done using a real CA):

```
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days validity
 -CAcreateserial -passin pass:ca-password
```

where:

- *ca-cert*: the certificate of the CA
- *ca-key*: the private key of the CA
- *cert-signed*: the signed certificate of the server
- *ca-password*: the passphrase of the CA

3. Import both the certificate of the CA and the signed certificate into the keystore:

```
keytool -keystore server.keystore.jks -alias CARoot -import -file ca-cert
keytool -keystore server.keystore.jks -alias localhost -import -file cert-signed
```

The following Bash script demonstrates the steps described above. One of the commands assumes a password of `SamplePassword123`, so either use that password or edit the command before running it.

```
#!/bin/bash
#Step 1
keytool -keystore server.keystore.jks -alias localhost -validity 365 -genkey
#Step 2
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
keytool -keystore server.truststore.jks -alias CARoot -import -file ca-cert
keytool -keystore client.truststore.jks -alias CARoot -import -file ca-cert
#Step 3
keytool -keystore server.keystore.jks -alias localhost -certreq -file cert-file
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days 365
-CAcreateserial -passin pass:SamplePassword123
keytool -keystore server.keystore.jks -alias CARoot -import -file ca-cert
keytool -keystore server.keystore.jks -alias localhost -import -file cert-signed
```

Step 4: Configuring Kafka Brokers

Kafka Brokers support listening for connections on multiple ports. If SSL is enabled for inter-broker communication (see below for how to enable it), both PLAINTEXT and SSL ports are required.

To configure the listeners from Cloudera Manager, perform the following steps:

1. In Cloudera Manager, go to **Kafka** > **Instances**.
2. Go to **Kafka Broker** > **Configurations**.
3. In the **Kafka Broker Advanced Configuration Snippet (Safety Valve) for Kafka Properties**, enter the following information:

```
listeners=PLAINTEXT://kafka-broker-host-name:9092,SSL://kafka-broker-host-name:9093
advertised.listeners=PLAINTEXT://kafka-broker-host-name:9092,SSL://kafka-broker-host-name:9093
```

where *kafka-broker-host-name* is the FQDN of the broker that you selected from the **Instances** page in Cloudera Manager. In the above sample configurations we used PLAINTEXT and SSL protocols for the SSL enabled brokers.

For information about other supported security protocols, see Using Kafka's Inter-Broker Security on page 34.

4. Repeat the previous step for each broker.

The `advertised.listeners` configuration is needed to connect the brokers from external clients.

5. Deploy the above client configurations and rolling restart the Kafka service from Cloudera Manager.

Kafka CSD auto-generates listeners for Kafka brokers, depending on your SSL and Kerberos configuration. To enable SSL for Kafka installations, do the following:

1. Turn on SSL for the Kafka service by turning on the `ssl_enabled` configuration for the Kafka CSD.
2. Set `security.inter.broker.protocol` as SSL, if Kerberos is disabled; otherwise, set it as SASL_SSL.

The following SSL configurations are required on each broker. Each of these values can be set in Cloudera Manager. Be sure to replace this example with the truststore password.

For instructions, see Changing the Configuration of a Service or Role Instance.

```
ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
ssl.keystore.password=SamplePassword123
ssl.key.password=SamplePassword123
ssl.truststore.location=/var/private/ssl/server.truststore.jks
ssl.truststore.password=SamplePassword123
```

Other configuration settings may also be needed, depending on your requirements:

- `ssl.client.auth=none`: Other options for client authentication are required, or requested, where clients without certificates can still connect. The use of requested is discouraged, as it provides a false sense of security and misconfigured clients can still connect.
- `ssl.cipher.suites`: A cipher suite is a named combination of authentication, encryption, MAC, and a key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol. This list is empty by default.
- `ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1`: Provide a list of SSL protocols that your brokers accept from clients.
- `ssl.keystore.type=JKS`
- `ssl.truststore.type=JKS`

Communication between Kafka brokers defaults to `PLAINTEXT`. To enable secured communication, modify the broker properties file by adding `security.inter.broker.protocol=SSL`.

For a list of the supported communication protocols, see Using Kafka's Inter-Broker Security on page 34.

> **Note:** Due to import regulations in some countries, Oracle implementation of JCA limits the strength of cryptographic algorithms. If you need stronger algorithms, you must obtain the JCE Unlimited Strength Jurisdiction Policy Files and install them in the JDK/JRE as described in JCA Providers Documentation.

After SSL is configured your broker, logs should show an endpoint for SSL communication:

```
with addresses: PLAINTEXT -> EndPoint(192.168.1.1,9092,PLAINTEXT),SSL ->
EndPoint(192.168.1.1,9093,SSL)
```

You can also check the SSL communication to the broker by running the following command:

```
openssl s_client -debug -connect localhost:9093 -tls1
```

This check can indicate that the server keystore and truststore are set up properly.

> **Note:** `ssl.enabled.protocols` should include `TLSv1`.

The output of this command should show the server certificate:

```
-----BEGIN CERTIFICATE-----
{variable sized random bytes}
-----END CERTIFICATE-----
subject=/C=US/ST=CA/L=Palo Alto/O=org/OU=org/CN=Franz Kafka
issuer=/C=US/ST=CA/L=Palo Alto
/O=org/OU=org/CN=kafka/emailAddress=kafka@your-domain.com
```

If the certificate does not appear, or if there are any other error messages, your keystore is not set up properly.

Step 5: Configuring Kafka Clients

SSL is supported only for the new Kafka producer and consumer APIs. The configurations for SSL are the same for both the producer and consumer.

If client authentication is not required in the broker, the following example shows a minimal configuration:

```
security.protocol=SSL
ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks
ssl.truststore.password=SamplePassword123
```

If client authentication is required, a keystore must be created as in step 1, it needs to be signed by the CA as in step 3, and you must also configure the following properties:

```
ssl.keystore.location=/var/private/ssl/kafka.client.keystore.jks
ssl.keystore.password=SamplePassword123
ssl.key.password=SamplePassword123
```

Other configuration settings might also be needed, depending on your requirements and the broker configuration:

- `ssl.provider` (Optional). The name of the security provider used for SSL connections. Default is the default security provider of the JVM.
- `ssl.cipher.suites` (Optional). A cipher suite is a named combination of authentication, encryption, MAC, and a key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol.
- `ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1`. This property should list at least one of the protocols configured on the broker side.
- `ssl.truststore.type=JKS`
- `ssl.keystore.type=JKS`

Using Kafka's Inter-Broker Security

Kafka can expose multiple communication endpoints, each supporting a different protocol. Supporting multiple communication endpoints enables you to use different communication protocols for client-to-broker communications and broker-to-broker communications. Set the Kafka inter-broker communication protocol using the `security.inter.broker.protocol` property. Use this property primarily for the following scenarios:

- Enabling SSL encryption for client-broker communication but keeping broker-broker communication as `PLAINTEXT`. Because SSL has performance overhead, you might want to keep inter-broker communication as `PLAINTEXT` if your Kafka brokers are behind a firewall and not susceptible to network snooping.
- Migrating from a non-secure Kafka configuration to a secure Kafka configuration without requiring downtime. Use a rolling restart and keep `security.inter.broker.protocol` set to a protocol that is supported by all brokers until all brokers are updated to support the new protocol.

For example, if you have a Kafka cluster that needs to be configured to enable Kerberos without downtime, follow these steps:

1. Set `security.inter.broker.protocol` to PLAINTEXT.
2. Update the Kafka service configuration to enable Kerberos.
3. Perform a rolling restart.
4. Set `security.inter.broker.protocol` to SASL_PLAINTEXT.

Kafka 2.0 and higher supports the combinations of protocols listed here.

	SSL	Kerberos
PLAINTEXT	No	No
SSL	Yes	No
SASL_PLAINTEXT	No	Yes
SASL_SSL	Yes	Yes

These protocols can be defined for broker-to-client interaction and for broker-to-broker interaction. The property `security.inter.broker.protocol` allows the broker-to-broker communication protocol to be different than the broker-to-client protocol, allowing rolling upgrades from non-secure to secure clusters. In most cases, set `security.inter.broker.protocol` to the protocol you are using for broker-to-client communication. Set `security.inter.broker.protocol` to a protocol different than the broker-to-client protocol only when you are performing a rolling upgrade from a non-secure to a secure Kafka cluster.

Enabling Kerberos Authentication

Apache Kafka supports Kerberos authentication, but it is supported only for the new Kafka Producer and Consumer APIs.

If you already have a Kerberos server, you can add Kafka to your current configuration. If you do not have a Kerberos server, install it before proceeding. See Enabling Kerberos Authentication for CDH.

If you already have configured the mapping from Kerberos principals to short names using the `hadoop.security.auth_to_local` HDFS configuration property, configure the same rules for Kafka by adding the `sasl.kerberos.principal.to.local.rules` property to the Advanced Configuration Snippet for Kafka Broker Advanced Configuration Snippet using Cloudera Manager. Specify the rules as a comma separated list.

To enable Kerberos authentication for Kafka:

1. In Cloudera Manager, navigate to **Kafka** > **Configuration**.
2. Set **SSL Client Authentication** to none.
3. Set **Inter Broker Protocol** to SASL_PLAINTEXT.
4. Click **Save Changes**.
5. Restart the Kafka service (**Action** > **Restart**).
6. Make sure that `listeners = SASL_PLAINTEXT` is present in the Kafka broker logs, by default in `/var/log/kafka/server.log`.
7. Create a `jaas.conf` file with either cached credentials or keytabs.

   To use cached Kerberos credentials, where you use `kinit` first, use this configuration.

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useTicketCache=true;
};
```

If you use a keytab, use this configuration. To generate keytabs, see Step 6: Get or Create a Kerberos Principal for Each User Account).

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
keyTab="/etc/security/keytabs/mykafkaclient.keytab"
principal="mykafkaclient/clients.hostname.com@EXAMPLE.COM";
};
```

8. Create the `client.properties` file containing the following properties.

```
security.protocol=SASL_PLAINTEXT
sasl.kerberos.service.name=kafka
```

9. Test with the Kafka console producer and consumer.

   To obtain a Kerberos ticket-granting ticket (TGT):

```
kinit user
```

10. Verify that your topic exists.

    This does not use security features, but it is a best practice.

```
kafka-topics --list --zookeeper zkhost:2181
```

11. Verify that the `jaas.conf` file is used by setting the environment.

```
export KAFKA_OPTS="-Djava.security.auth.login.config=/home/user/jaas.conf"
```

12. Run a Kafka console producer.

```
kafka-console-producer --broker-list anybroker:9092 --topic test1 --producer.config
client.properties
```

13. Run a Kafka console consumer.

```
kafka-console-consumer --new-consumer --topic test1 --from-beginning --bootstrap-server
 anybroker:9092 --consumer.config client.properties
```

Enabling Encryption at Rest

Data encryption is increasingly recognized as an optimal method for protecting data at rest. You can encrypt Kafka data using Cloudera Navigator Encrypt.

Perform the following steps to encrypt Kafka data that is not in active use.

1. Stop the Kafka service.
2. Archive the Kafka data to an alternate location, using TAR or another archive tool.
3. Unmount the affected drives.
4. Install and configure Navigator Encrypt.

   See Installing Cloudera Navigator Encrypt.

5. Expand the TAR archive into the encrypted directories.

6. Restart the Kafka service.

Topic Authorization with Kerberos and Sentry

Apache Sentry includes a Kafka binding you can use to enable authorization in Kafka with Sentry. For more information, see Authorization With Apache Sentry.

Configuring Kafka to Use Sentry Authorization

The following steps describe how to configure Kafka to use Sentry authorization. These steps assume you have installed Kafka and Sentry on your cluster.

Sentry requires that your cluster include HDFS. After you install and start Sentry with the correct configuration, you can stop the HDFS service. For more information, see Installing and Upgrading the Sentry Service.

> **Note:** Cloudera's distribution of Kafka can make use of LDAP-based user groups when the LDAP directory is synchronized to Linux via tools such as SSSD. CDK does not support direct integration with LDAP, either through direct Kafka's LDAP authentication, or via Hadoop's group mapping (when `hadoop.group.mapping` is set to LdapGroupMapping). For more information, see Configuring LDAP Group Mappings.

To configure Sentry authentication for Kafka:

1. In Cloudera Manager, go to **Kafka** > **Configuration**.
2. Select **Enable Kerberos Authentication**.
3. Select a Sentry service in the Kafka service configuration.
4. Add superusers.

   Superusers can perform any action on any resource in the Kafka cluster. The `kafka` user is added as a superuser by default. Superuser requests are authorized without going through Sentry, which provides enhanced performance.

5. Select **Enable Sentry Privileges Caching** to enhance performance.
6. Restart the Sentry services.

Authorizable Resources

Authorizable resources are resources or entities in a Kafka cluster that require special permissions for a user to be able to perform actions on them. Kafka has four authorizable resources.

- **Cluster:** controls who can perform cluster-level operations such as creating or deleting a topic. This resource can only have one value, `kafka-cluster`, as one Kafka cluster cannot have more than one cluster resource.
- **Topic:** controls who can perform topic-level operations such as producing and consuming topics. Its value must match exactly the topic name in the Kafka cluster.

  With CDH 5.15.0 and CDK 3.1 and later, wildcards (*) can be used to refer to any topic in the privilege.

- **Consumergroup:** controls who can perform consumergroup-level operations such as joining or describing a consumergroup. Its value must exactly match the `group.id` of a consumergroup.

  With CDH 5.14.1 and later, you can use a wildcard (*) to refer to any consumer groups in the privilege. This resource is useful when used with Spark Streaming, where a generated `group.id` may be needed.

- **Host:** controls from where specific operations can be performed. Think of this as a way to achieve IP filtering in Kafka. You can set the value of this resource to the wildcard (*), which represents all hosts.

  > **Note:** Only IP addresses should be specified in the host component of Kafka Sentry privileges, hostnames are not supported.

Authorized Actions

You can perform multiple actions on each resource. The following operations are supported by Kafka, though not all actions are valid on all resources.

- `ALL` is a wildcard action, and represents all possible actions on a resource.
- `read`
- `write`
- `create`
- `delete`
- `alter`
- `describe`
- `clusteraction`

Authorizing Privileges

Privileges define what actions are allowed on a resource. A privilege is represented as a string in Sentry. The following rules apply to a valid privilege.

- Can have at most one Host resource. If you do not specify a Host resource in your privilege string, `Host=*` is assumed.
- Must have exactly one non-Host resource.
- Must have exactly one action specified at the end of the privilege string.

For example, the following are valid privilege strings:

```
Host=*->Topic=myTopic->action=ALL
Topic=test->action=ALL
```

Granting Privileges to a Role

The following examples grant privileges to the role test, so that users in `testGroup` can create a topic named `testTopic` and produce to it.

The user executing these commands must be added to the Sentry parameter `sentry.service.allow.connect` and also be a member of a group defined in `sentry.service.admin.group`.

Before you can assign the test role, you must first create it. To create the `test` role:

```
kafka-sentry -cr -r test
```

To confirm that the role was created, list the roles:

```
kafka-sentry -lr
```

If Sentry privileges caching is enabled, as recommended, the new privileges you assign take some time to appear in the system. The time is the time-to-live interval of the Sentry privileges cache, which is set using `sentry.kafka.caching.ttl.ms`. By default, this interval is 30 seconds. For test clusters, it is beneficial to have changes appear within the system as fast as possible, therefore, Cloudera recommends that you either use a lower time interval, or disable caching with `sentry.kafka.caching.enable`.

1. Allow users in `testGroup` to write to `testTopic` from `localhost`, which allows users to produce to `testTopic`. Users need both `write` and `describe` permissions.

```
kafka-sentry -gpr -r test -p "Host=127.0.0.1->Topic=testTopic->action=write"
kafka-sentry -gpr -r test -p "Host=127.0.0.1->Topic=testTopic->action=describe"
```

2. Assign the test role to the group testGroup:

```
kafka-sentry -arg -r test -g testGroup
```

3. Verify that the test role is part of the group testGroup:

```
kafka-sentry -lr -g testGroup
```

4. Create testTopic.

```
$ kafka-topics --create --zookeeper localhost:2181 \
  --replication-factor 1 \
  --partitions 1 --topic testTopic
kafka-topics --list --zookeeper localhost:2181 testTopic
```

Now you can produce to and consume from the Kafka cluster.

1. Produce to `testTopic`.

   Note that you have to pass a configuration file, `producer.properties`, with information on JAAS configuration and other Kerberos authentication related information. See SASL Configuration for Kafka Clients.

```
$ kafka-console-producer --broker-list localhost:9092 \
  --topic testTopic --producer.config producer.properties
  This is a message
  This is another message
```

2. Grant the `create` privilege to the test role.

```
$ kafka-sentry -gpr -r test -p
"Host=127.0.0.1->Cluster=kafka-cluster->action=create"
```

3. Allow users in `testGroup` to describe `testTopic` from `localhost`, which the user creates and uses.

```
$ kafka-sentry -gpr -r test -p
"Host=127.0.0.1->Topic=testTopic->action=describe"
```

4. Grant the describe privilege to the test role.

```
$ kafka-sentry -gpr -r test -p
"Host=127.0.0.1->Consumergroup=testconsumergroup->action=describe"
```

5. Allow users in `testGroup` to read from a consumer group, `testconsumergroup`, that it will start and join.

```
$ kafka-sentry -gpr -r test -p
"Host=127.0.0.1->Consumergroup=testconsumergroup->action=read"
```

6. Allow users in `testGroup` to read from `testTopic` from `localhost` and to consume from `testTopic`.

```
$ kafka-sentry -gpr -r test -p
"Host=127.0.0.1->Topic=testTopic->action=read"
```

7. Consume from `testTopic`.

   Note that you have to pass a configuration file, `consumer.properties`, with information on JAAS configuration and other Kerberos authentication related information. The configuration file must also specify `group.id` as `testconsumergroup`.

```
kafka-console-consumer --new-consumer --topic testTopic \
  --from-beginning --bootstrap-server anybroker-host:9092 \
  --consumer.config consumer.properties
```

```
This is a message
This is another message
```

Troubleshooting Kafka with Sentry

If Kafka requests are failing due to authorization, the following steps can provide insight into the error:

- Make sure you have run `kinit` as a user who has privileges to perform an operation.
- Identify which broker is hosting the leader of the partition you are trying to produce to or consume from, as this leader is going to authorize your request against Sentry. One easy way of debugging is to just have one Kafka broker. Change log level of the Kafka broker by adding the following entry to the Kafka Broker in Logging Advanced Configuration Snippet (Safety Valve) and restart the broker:

```
log4j.logger.org.apache.sentry=DEBUG
```

  Setting just Sentry to DEBUG mode avoids the debug output from undesired dependencies, such as Jetty.

- Run the Kafka client or Kafka CLI with the required arguments and capture the Kafka log, which should be similar to:

```
/var/log/kafka/kafka-broker-host-name.log
```

- Look for the following information in the filtered logs:
  - Groups that the Kafka client user or CLI is running as.
  - Required privileges for the operation.
  - Retrieved privileges from Sentry.
  - Required and retrieved privileges comparison result.

This log information can provide insight into which privilege is not assigned to a user, causing a particular operation to fail.

Managing Multiple Kafka Versions

Kafka Feature Support in Cloudera Manager and CDH

Using the latest features of Kafka sometimes requires a newer version of Cloudera Manager and/or CDH that supports the feature. For the purpose of understanding upgrades, the table below includes several earlier versions where Kafka was known as CDK Powered by Apache Kafka (CDK in the table).

Table 1: Kafka and CM/CDH Supported Versions Matrix

CDH Kafka Version	Apache Kafka Version	Minimum Supported Version			
		Cloudera Manager	CDH (No Security)	CDH (with Sentry)	Kafka Feature Notes
CDH 6.2.0	2.1.0	CM 6.2.0	CDH 6.2.0	CDH 6.2.0	
CDH 6.1.0	2.0.0	CM 6.1.0	CDH 6.1.0	CDH 6.1.0	
CDH 6.0.0	1.0.1	CM 6.0.0	CDH 6.0.0	CDH 6.0.0	
CDK 3.1.0	1.0.1	CM 5.13.0	CDH 5.13.0	CDH 5.13.0	Sentry-HA supported
CDK 3.0.0	0.11.0	CM 5.13.0	CDH 5.13.0	CDH 5.13.0	Sentry-HA not supported
CDK 2.2	0.10.2	CM 5.9.0	CDH 5.4.0	CDH 5.9.0	

CDH Kafka Version	Apache Kafka Version	Minimum Supported Version			
		Cloudera Manager	CDH (No Security)	CDH (with Sentry)	Kafka Feature Notes
CDK 2.1	0.10.0	CM 5.9.0	CDH 5.4.0	CDH 5.9.0	Sentry Authorization
CDK 2.0	0.9.0	CM5.5.3	CDH 5.4.0	CDH 5.4.0	Enhanced Security

Client/Broker Compatibility Across Kafka Versions

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.

Upgrading your Kafka Cluster

At some point, you will want to upgrade your Kafka cluster. Or in some cases, when you upgrade to a newer CDH distribution, then Kafka will be upgraded along with the full CDH upgrade. In either case, you may wish to read the sections below before upgrading.

General Upgrade Information

Previously, Cloudera distributed Kafka as a parcel (CDK) separate from CDH. Installation of the separate Kafka required Cloudera Manager 5.4 or higher. For a list of available parcels and packages, see CDK Powered By Apache Kafka® Version and Packaging Information Kafka bundled along with CDH.

As of CDH 6, Cloudera distributes Kafka bundled along with CDH. Kafka is in the parcel itself. Installation requires Cloudera Manager 6.0 or higher. For installation instructions for Kafka using Cloudera Manager, see Cloudera Installation Guide.

Cloudera recommends that you deploy Kafka on dedicated hosts that are not used for other cluster roles.

Important: You cannot install an old Kafka parcel on a new CDH 6.x cluster.

Upgrading Kafka from CDH 6.0.0 to other CDH 6 versions

To ensure there is no downtime during an upgrade, these instructions describe performing a rolling upgrade.

Before upgrading, ensure that you set `inter.broker.protocol.version` and `log.message.format.version` to the current Kafka version (see Table 1: Kafka and CM/CDH Supported Versions Matrix on page 40), and then unset them after the upgrade. This is a good practice because the newer broker versions can write log entries that the older brokers cannot read. If you need to rollback to the older version, and you have not set `inter.broker.protocol.version` and `log.message.format.version`, data loss can occur.

From Cloudera Manager on the cluster to upgrade:

1. Explicitly set the Kafka protocol version to match what's being used currently among the brokers and clients.

   Update `server.properties` on all brokers as follows:

   a. Choose the Kafka service.
   b. Click **Configuration**.
   c. Use the Search field to find the **Kafka Broker Advanced Configuration Snippet (Safety Valve)** configuration property.
   d. Add the following properties to the safety valve:

```
inter.broker.protocol.version = current_Kafka_version
log.message.format.version = current_Kafka_version
```

   Make sure you enter full Kafka version numbers with three values, such as 0.10.0. Otherwise, you'll see an error similar to the following:

```
2018-06-14 14:25:47,818 FATAL kafka.Kafka$:
java.lang.IllegalArgumentException: Version `0.10` is not a valid version
        at kafka.api.ApiVersion$$anonfun$apply$1.apply(ApiVersion.scala:72)
        at kafka.api.ApiVersion$$anonfun$apply$1.apply(ApiVersion.scala:72)
        at scala.collection.MapLike$class.getOrElse(MapLike.scala:128)
```

   e. Save your changes. The information is automatically copied to each broker.

2. Upgrade CDH. See Upgrading the CDH Cluster.

   Do not restart the Kafka service, select **Activate Only** and click **OK**.

3. Perform a rolling restart.

   Select **Rolling Restart** or **Restart** based on the downtime that you can afford.

At this point, the brokers are running in compatibility mode with older clients. It can run in this mode indefinitely. If you do upgrade clients, after all clients are upgraded, remove the Safety Valve properties and restart the cluster.

Upstream Upgrade Instructions

The table below links to the upstream Apache Kafka documentation for detailed upgrade instructions. It is recommended that you read the instructions for your specific upgrade to identify any additional steps that apply to your specific upgrade path.

Table 2: Version-Specific Upgrade Instructions

CDH Kafka Version	Upstream Kafka Version	Detailed Upstream Instructions
CDH 6.2.0	2.1.0	Upgrading from 0.8.x through 2.0.0.x to 2.1.0
CDH 6.1.0	2.0.0	Upgrading from 0.8.x through 1.1.x to 2.0.0
CDH 6.0.0	1.0.1	Upgrading from 0.8.x through 0.11.0.x to 1.0.0
Kafka 3.1.0	1.0.1	Upgrading from 0.8.x through 0.11.0.x to 1.0.0

CDH Kafka Version	Upstream Kafka Version	Detailed Upstream Instructions
Kafka 3.0.0	0.11.0	Upgrading from 0.8.x through 0.10.2.x to 0.11.0.0
Kafka 2.2	0.10.2	Upgrading from 0.8.x through 0.10.1.x to 0.10.2.0
Kafka 2.1	0.10.0	Upgrading from 0.8.x through 0.9.x.to 0.10.0.0
Kafka 2.0	0.9.0	Upgrading from 0.8.0 through 0.8.2.x to 0.9.0.0

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.

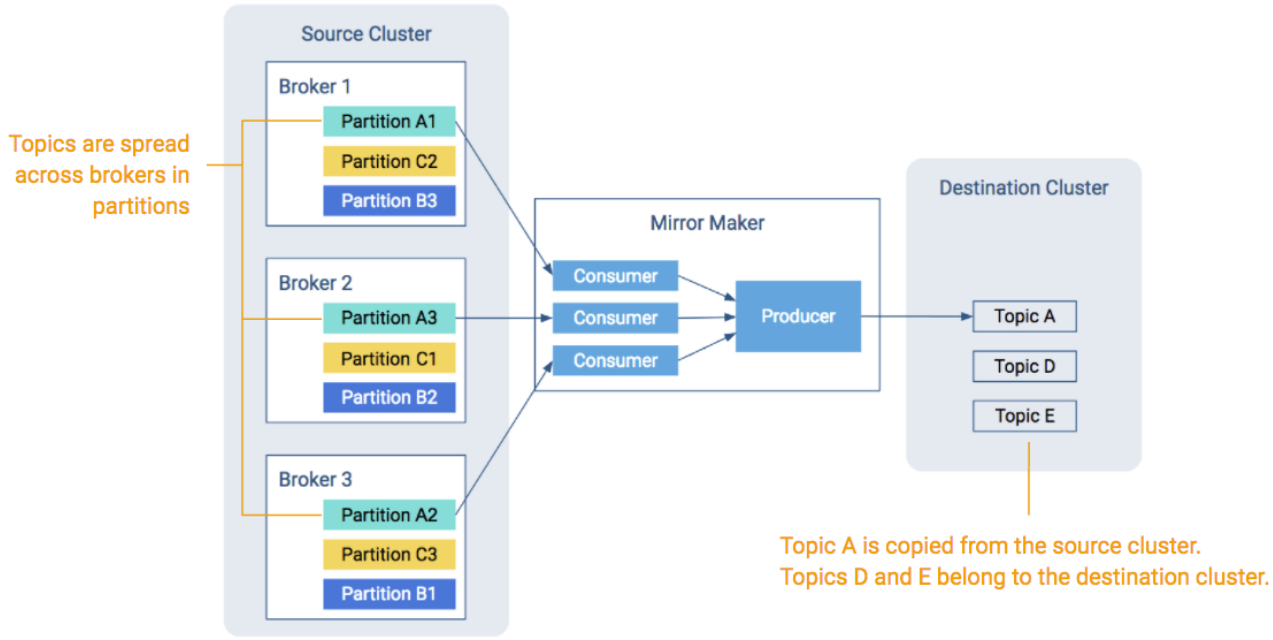


Figure 18: Mirror Maker Makes Topics Available on Multiple Clusters

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.

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/Broker Compatibility Across Kafka Versions on page 41 for more details about what it means to be "compatible."

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 use the `kafka-configs` command to set the property `auto.create.topics.enable=true`. See Kafka Administration Using Command Line Tools on page 63.

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.

Setting up Mirror Maker in Cloudera Manager

Where Cloudera Manager is managing the destination cluster:

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 whitelist is required.

7. Fill out the TLS/SSL sections if security needs to be enabled.
8. Start the Mirror Maker instance.

Settings to Avoid Data Loss

The **Avoid Data Loss** option from earlier releases has been removed in favor of automatically setting the following properties. Also 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`

Setting up an End-to-End Data Streaming Pipeline

Data Streaming Pipeline

The data streaming pipeline as shown here is the most common usage of Kafka.

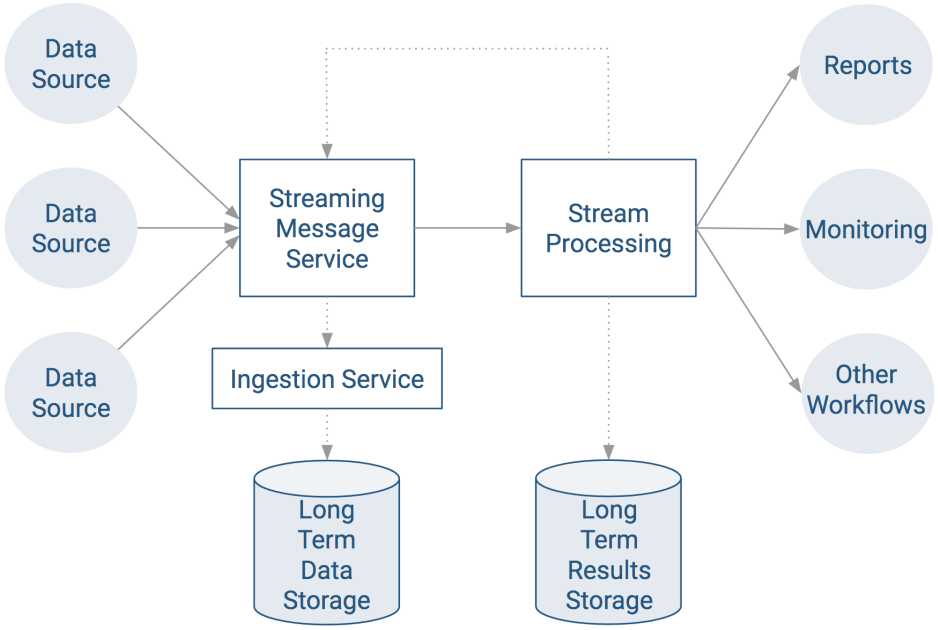


Figure 19: Data Streaming Pipeline Architecture

Some things to note about the data streaming pipeline model:

- Most systems have multiple data sources sending data over the Internet, such as per store or per device.
- The ingestion service usually saves older data to some form of long-term data storage.
- The stream processing service can perform near real-time computation on the data extracted from the message service, such as processing transactions, detecting fraud, or alerting systems.
- The results of stream processing can be sent directly to another service (such as for reporting) or can be streamed back into Kafka for one or more other services to do further real time processing.

The following sections show how other components in CDH map into the data streaming model.

Ingest Using Kafka with Apache Flume

Apache Flume is commonly used to collect Kafka topics into a long-term data store.

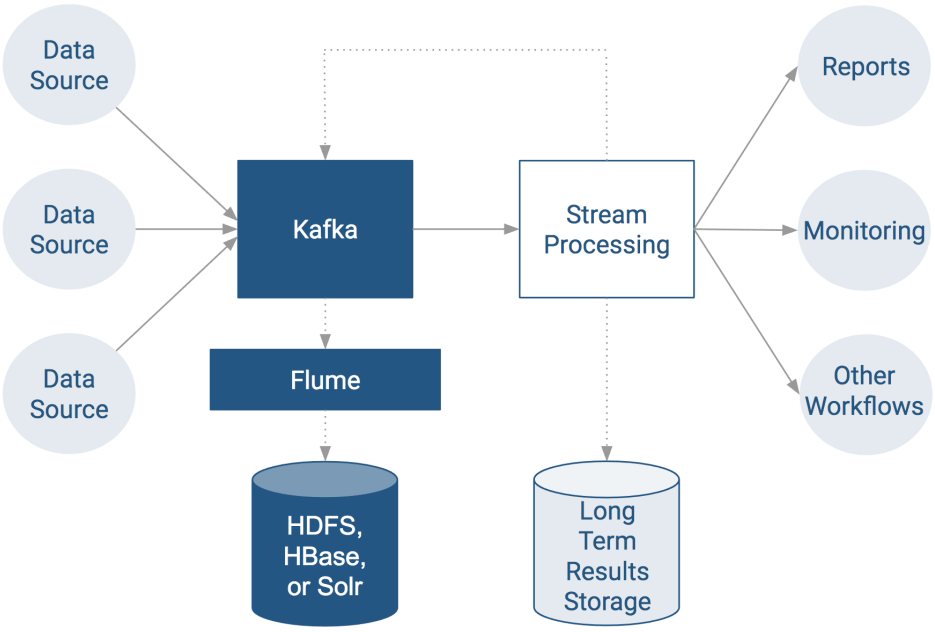


Figure 20: Collecting Kafka Topics using Flume

Note: Do not configure a Kafka source to send data to a Kafka sink. If you do, the Kafka source sets the topic in the event header, overriding the sink configuration and creating an infinite loop, sending messages back and forth between the source and sink. If you need to use both a Kafka source and a sink, use an interceptor to modify the event header and set a different topic.

For information on configuring Kafka to securely communicate with Flume, see Configuring Flume Security with Kafka. The following sections describe how to configure Kafka sub-components for directing topics to long-term storage:

Sources

Use the Kafka source to stream data in Kafka topics to Hadoop. The Kafka source can be combined with any Flume sink, making it easy to write Kafka data to HDFS, HBase, and Solr.

The following Flume configuration example uses a Kafka source to send data to an HDFS sink:

```
tier1.sources  = source1
tier1.channels = channel1
tier1.sinks = sink1

tier1.sources.source1.type = org.apache.flume.source.kafka.KafkaSource
tier1.sources.source1.zookeeperConnect = zk01.example.com:2181
tier1.sources.source1.topic = weblogs
tier1.sources.source1.groupId = flume
tier1.sources.source1.channels = channel1
tier1.sources.source1.interceptors = i1
tier1.sources.source1.interceptors.i1.type = timestamp
tier1.sources.source1.kafka.consumer.timeout.ms = 100

tier1.channels.channel1.type = memory
tier1.channels.channel1.capacity = 10000
tier1.channels.channel1.transactionCapacity = 1000

tier1.sinks.sink1.type = hdfs
tier1.sinks.sink1.hdfs.path = /tmp/kafka/%{topic}/%y-%m-%d
tier1.sinks.sink1.hdfs.rollInterval = 5
tier1.sinks.sink1.hdfs.rollSize = 0
tier1.sinks.sink1.hdfs.rollCount = 0
tier1.sinks.sink1.hdfs.fileType = DataStream
```

```
tier1.sinks.sink1.channel = channel1
```

For higher throughput, configure multiple Kafka sources to read from the same topic. If you configure all the sources with the same groupID, and the topic contains multiple partitions, each source reads data from a different set of partitions, improving the ingest rate.

The following table describes parameters that the Kafka source supports. Required properties are listed in **bold**.

Property Name	Default Value	Description
`type`		Must be set to `org.apache.flume.source.kafka.KafkaSource`.
`zookeeperConnect`		The URI of the ZooKeeper server or quorum used by Kafka. This can be a single host (for example, `zk01.example.com:2181`) or a comma-separated list of hosts in a ZooKeeper quorum (for example, `zk01.example.com:2181,zk02.example.com:2181, zk03.example.com:2181`).
`topic`		The Kafka topic from which this source reads messages. Flume supports only one topic per source.
`groupID`	flume	The unique identifier of the Kafka consumer group. Set the same groupID in all sources to indicate that they belong to the same consumer group.
`batchSize`	1000	The maximum number of messages that can be written to a channel in a single batch.
`batchDurationMillis`	1000	The maximum time (in ms) before a batch is written to the channel. The batch is written when the batchSize limit or batchDurationMillis limit is reached, whichever comes first.
Other properties supported by the Kafka consumer		Used to configure the Kafka consumer used by the Kafka source. You can use any consumer properties supported by Kafka. Prepend the consumer property name with the prefix `kafka.` (for example, `kafka.fetch.min.bytes`). See the Apache Kafka documentation topic Consumer Configs for the full list of Kafka consumer properties.

Source Tuning Notes

The Kafka source overrides two Kafka consumer parameters:

1. `auto.commit.enable` is set to `false` by the source, committing every batch. For improved performance, set this parameter to `true` using the `kafka.auto.commit.enable` setting. Note that this change can lead to data loss if the source goes down before committing.
2. `consumer.timeout.ms` is set to 10, so when Flume polls Kafka for new data, it waits no more than 10 ms for the data to be available. Setting this parameter to a higher value can reduce CPU utilization due to less frequent polling, but the trade-off is that it introduces latency in writing batches to the channel.

Kafka Sinks

Use the Kafka sink to send data to Kafka from a Flume source. You can use the Kafka sink in addition to Flume sinks, such as HBase or HDFS.

The following Flume configuration example uses a Kafka sink with an exec source:

```
tier1.sources  = source1
tier1.channels = channel1
tier1.sinks = sink1

tier1.sources.source1.type = exec
tier1.sources.source1.command = /usr/bin/vmstat 1
tier1.sources.source1.channels = channel1

tier1.channels.channel1.type = memory
tier1.channels.channel1.capacity = 10000
tier1.channels.channel1.transactionCapacity = 1000

tier1.sinks.sink1.type = org.apache.flume.sink.kafka.KafkaSink
tier1.sinks.sink1.topic = sink1
tier1.sinks.sink1.brokerList = kafka01.example.com:9092,kafka02.example.com:9092
tier1.sinks.sink1.channel = channel1
tier1.sinks.sink1.batchSize = 20
```

The following table describes parameters the Kafka sink supports. Required properties are listed in **bold**.

Property Name	Default Value	Description
`type`		Must be set to `org.apache.flume.sink.kafka.KafkaSink`.
`brokerList`		The brokers the Kafka sink uses to discover topic partitions, formatted as a comma-separated list of *hostname*:*port* entries. You do not need to specify the entire list of brokers, but you specify at least two for high availability.
`topic`	default-flume-topic	The Kafka topic to which messages are published by default. If the event header contains a topic field, the event is published to the designated topic, overriding the configured topic.
`batchSize`	100	The number of messages to process in a single batch. Specifying a larger batchSize can improve throughput and increase latency.
`request.required.acks`	0	The number of replicas that must acknowledge a message before it is written successfully. Possible values are:

Property Name	Default Value	Description
		`0` do not wait for an acknowledgment `1` wait for the leader to acknowledge only `-1` wait for all replicas to acknowledge To avoid potential loss of data in case of a leader failure, set this to `-1`.
Other properties supported by the Kafka producer		Used to configure the Kafka producer used by the Kafka sink. You can use any producer properties supported by Kafka. Prepend the producer property name with the prefix `kafka` (for example, `kafka.compression.codec`). See the Apache Kafka documentation topic Producer Configs for the full list of Kafka producer properties.

The Kafka sink uses the `topic` and `key` properties from the FlumeEvent headers to determine where to send events in Kafka. If the header contains the `topic` property, that event is sent to the designated topic, overriding the configured topic. If the header contains the `key` property, that key is used to partition events within the topic. Events with the same key are sent to the same partition. If the `key` parameter is not specified, events are distributed randomly to partitions. Use these properties to control the topics and partitions to which events are sent through the Flume source or interceptor.

Kafka Channels

CDH includes a Kafka channel to Flume in addition to the existing memory and file channels. You can use the Kafka channel:

- To write to Hadoop directly from Kafka without using a source.
- To write to Kafka directly from Flume sources without additional buffering.
- As a reliable and highly available channel for any source/sink combination.

The following Flume configuration uses a Kafka channel with an exec source and HDFS sink:

```
tier1.sources = source1
tier1.channels = channel1
tier1.sinks = sink1

tier1.sources.source1.type = exec
tier1.sources.source1.command = /usr/bin/vmstat 1
tier1.sources.source1.channels = channel1

tier1.channels.channel1.type = org.apache.flume.channel.kafka.KafkaChannel
tier1.channels.channel1.capacity = 10000
tier1.channels.channel1.zookeeperConnect = zk01.example.com:2181
tier1.channels.channel1.parseAsFlumeEvent = false
tier1.channels.channel1.topic = channel2
tier1.channels.channel1.consumer.group.id = channel2-grp
tier1.channels.channel1.auto.offset.reset = earliest
tier1.channels.channel1.kafka.bootstrap.servers =
```

```
kafka02.example.com:9092,kafka03.example.com:9092
tier1.channels.channel1.transactionCapacity = 1000
tier1.channels.channel1.kafka.consumer.max.partition.fetch.bytes=2097152

tier1.sinks.sink1.type = hdfs
tier1.sinks.sink1.hdfs.path = /tmp/kafka/channel
tier1.sinks.sink1.hdfs.rollInterval = 5
tier1.sinks.sink1.hdfs.rollSize = 0
tier1.sinks.sink1.hdfs.rollCount = 0
tier1.sinks.sink1.hdfs.fileType = DataStream
tier1.sinks.sink1.channel = channel1
```

The following table describes parameters the Kafka channel supports. Required properties are listed in **bold**.

Property Name	Default Value	Description
`type`		Must be set to `org.apache.flume.channel.kafka.KafkaChannel`.
`brokerList`		The brokers the Kafka channel uses to discover topic partitions, formatted as a comma-separated list of *hostname*:*port* entries. You do not need to specify the entire list of brokers, but you should specify at least two for high availability.
`zookeeperConnect`		The URI of the ZooKeeper server or quorum used by Kafka. This can be a single host (for example, `zk01.example.com:2181`) or a comma-separated list of hosts in a ZooKeeper quorum (for example, `zk01.example.com:2181,zk02.example.com:2181, zk03.example.com:2181`).
`topic`	flume-channel	The Kafka topic the channel will use.
`groupID`	flume	The unique identifier of the Kafka consumer group the channel uses to register with Kafka.
`parseAsFlumeEvent`	true	Set to true if a Flume source is writing to the channel and expects AvroDataums with the FlumeEvent schema (`org.apache.flume.source.avro.AvroFlumeEvent`) in the channel. Set to false if other producers are writing to the topic that the channel is using.
`auto.offset.reset`	latest	What to do when there is no initial offset in Kafka or if the current offset does not exist on the server (for example, because the data is deleted). • `earliest`: automatically reset the offset to the earliest offset

Property Name	Default Value	Description
		• `latest`: automatically reset the offset to the latest offset • none: throw exception to the consumer if no previous offset is found for the consumer's group • anything else: throw exception to the consumer.
`kafka.consumer.timeout.ms`	100	Polling interval when writing to the sink.
`consumer.max.partition.fetch.bytes`	1048576	The maximum amount of data per-partition the server will return.
Other properties supported by the Kafka producer		Used to configure the Kafka producer. You can use any producer properties supported by Kafka. Prepend the producer property name with the prefix `kafka.` (for example, `kafka.compression.codec`). See the Apache Kafka documentation topic Producer Configs for the full list of Kafka producer properties.

CDH Flume Kafka Compatibility

The section Client/Broker Compatibility Across Kafka Versions on page 41 covered the basics of Kafka client/broker compatibility. Flume has an embedded Kafka client which it uses to talk to Kafka clusters. Since the generally accepted practice is to have the broker running the same or newer version as the client, a CDH Flume version requires being matched to a minimum Kafka version. This is illustrated in the table below.

Table 3: Flume Embedded Client and Kafka Compatibility

CDH Flume Version	Embedded Kafka Client Version	Minimum Supported CDH Kafka Version (Remote or Local)
CDH 6.0.0	1.0.1	CDH 6.0.0
CDH 5.14.x	0.10.2-kafka-2.2.0	Kafka 2.2
CDH 5.13.x	0.9.0-kafka-2.0.2	Kafka 2.0
CDH 5.12.x	0.9.0-kafka-2.0.2	Kafka 2.0
CDH 5.11.x	0.9.0-kafka-2.0.2	Kafka 2.0
CDH 5.10.x	0.9.0-kafka-2.0.2	Kafka 2.0
CDH 5.9.x	0.9.0-kafka-2.0.2	Kafka 2.0
CDH 5.8.x	0.9.0-kafka-2.0.0	Kafka 2.0
CDH 5.7.x	0.9.0-kafka-2.0.0	Kafka 2.0

Securing Flume with Kafka

When using Flume with a secured Kafka service, you can use Cloudera Manager to generate security related Flume agent configuration.

In Cloudera Manager, on the Flume **Configuration** page, select the Kafka service you want to connect to. This generates the following files:

- `flume.keytab`
- `jaas.conf`

It also generates security protocol and Kerberos service name properties for the Flume agent configuration. If TLS/SSL is also configured for Kafka brokers, the setting also adds SSL truststore properties to the beginning of the Flume agent configuration.

Review the deployed agent configuration and if the defaults do not match your environment (such as the truststore password), you can override the settings by adding the same property to the agent configuration.

Using Kafka with Apache Spark Streaming for Stream Processing

For real-time stream computation, Apache Spark is the tool of choice in CDH.

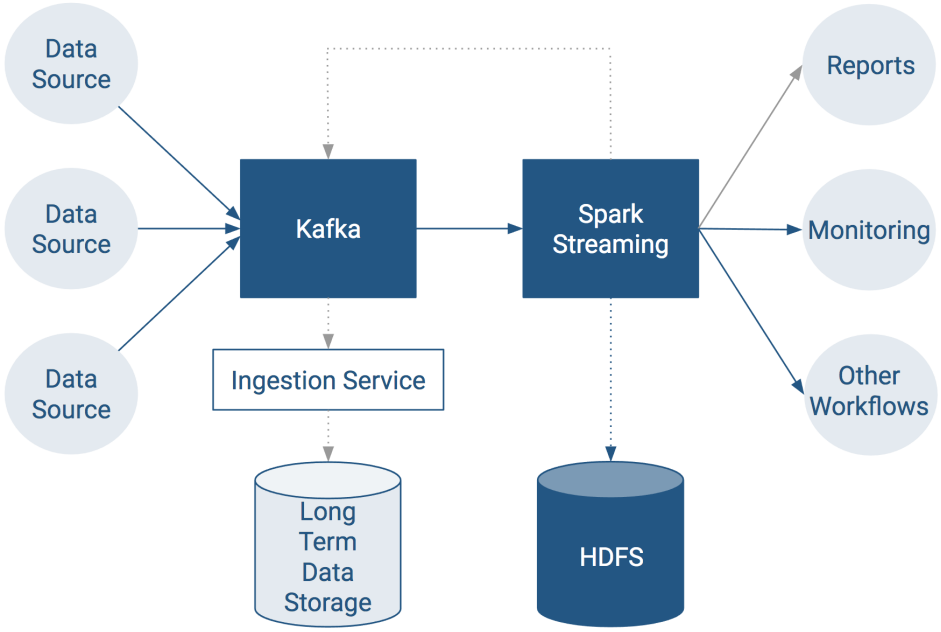


Figure 21: Data Streaming Pipeline with Spark

CDH Spark/Kafka Compatibility

The section Client/Broker Compatibility Across Kafka Versions on page 41 covered the basics of Kafka client/broker compatibility. Spark maintains two embedded Kafka clients and can be configured to use either one. This table in the Apache Spark documentation illustrates the two clients that are available.

For information on how to configure Spark Streaming to receive data from Kafka, refer to the Kafka version you are using in the following table.

CDH Spark Version	Embedded Kafka Client (Choose based on Kafka cluster)	Minimum Apache Kafka Version	Minimum CDH Kafka Version (Remote or Local)	Upstream Integration Guide	API Stability
CDH 6.0	spark-streaming-kafka-0-10	0.10.0	Kafka 2.1	Spark 2.2 + Kafka 0.10	Stable
CDH 6.0	spark-streaming-kafka-0-8	0.8.2.1	Kafka 2.0	Spark 2.2 + Kafka 0.8	Deprecated
Spark 2.2	spark-streaming-kafka-0-10	0.10.0	Kafka 2.1	Spark 2.2 + Kafka 0.10	Experimental
Spark 2.2	spark-streaming-kafka-0-8	0.8.2.1	Kafka 2.0	Spark 2.2 + Kafka 0.8	Stable

CDH Spark Version	Embedded Kafka Client (Choose based on Kafka cluster)	Minimum Apache Kafka Version	Minimum CDH Kafka Version (Remote or Local)	Upstream Integration Guide	API Stability
Spark 2.1	spark-streaming-kafka-0-10	0.10.0	Kafka 2.1	Spark 2.1 + Kafka 0.10	Experimental
Spark 2.1	spark-streaming-kafka-0-8	0.8.2.1	Kafka 2.0	Spark 2.1 + Kafka 0.8	Stable
Spark 2.0	spark-streaming-kafka-0-8	0.8.2.?	Kafka 2.0	Spark 2.0 + Kafka	Stable

Validating Kafka Integration with Spark Streaming

To validate your Kafka integration with Spark Streaming, run the `KafkaWordCount` example in Spark.

If you installed Spark using parcels, use the following command:

```
/opt/cloudera/parcels/CDH/lib/spark/bin/run-example streaming.KafkaWordCount zkQuorum group topics numThreads
```

If you installed Spark using packages, use the following command:

```
/usr/lib/spark/bin/run-example streaming.KafkaWordCount zkQuorum group topics numThreads
```

Replace the variables as follows:

- *zkQuorum*: ZooKeeper quorum URI used by Kafka For example:

```
zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181
```

- *group*: Consumer group used by the application.
- *topics*: Kafka topic containing the data for the application.
- *numThreads*: Number of consumer threads reading the data. If this is higher than the number of partitions in the Kafka topic, some threads will be idle.

Note: If multiple applications use the same group and topic, each application receives a subset of the data.

Securing Spark with Kafka

Using Spark Streaming with a Kafka service that's already secured requires configuration changes on the Spark side. You can find a nice description of the required changes in the `spark-dstream-secure-kafka-app` sample project on GitHub.

Developing Kafka Clients

Previously, examples were provided for producing messages to and consuming messages from a Kafka cluster using the command line. For most cases, running Kafka producers and consumers using shell scripts and Kafka's command line scripts cannot be used in practice. In those cases, native Kafka client development is the generally accepted option.

Simple Client Examples

Let's start with a simple working example of a producer/consumer program. This section includes the following code examples:

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.cloudera.kafkaexamples</groupId>
  <artifactId>kafka-examples</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>kafkadev</name>
  <url>http://maven.apache.org</url>
  <repositories>
    <repository>
      <id>cloudera</id>
      <url>https://repository.cloudera.com/artifactory/cloudera-repos/</url>
    </repository>
  </repositories>
  <dependencies>
    <dependency>
      <groupId>org.apache.kafka</groupId>
      <artifactId>kafka-clients</artifactId>
      <version>1.0.1-cdh6.0.0</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.7.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

SimpleProducer.java

The example includes Java properties for setting up the client identified in the comments; the functional parts of the code are in **bold**. This code is compatible with versions as old as the 0.9.0-kafka-2.0.0 version of Kafka.

```
package com.cloudera.kafkaexamples;

import java.util.Date;
import java.util.Properties;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;

public class SimpleProducer {
    public static void main(String[] args) {
        // Generate total consecutive events starting with ufoId
        long total = Long.parseLong("10");
        long ufoId = Math.round(Math.random() * Integer.MAX_VALUE);

        // Set up client Java properties
        Properties props = new Properties();
        props.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
                "host1:9092,host2:9092,host3:9092");
```

```
        props.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
                StringSerializer.class.getName());
        props.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
                StringSerializer.class.getName());
        props.setProperty(ProducerConfig.ACKS_CONFIG, "1");

        try (KafkaProducer<String, String> producer = new KafkaProducer<>(props)) {
            for (long i = 0; i < total; i++) {
                String key = Long.toString(ufoId++);
                long runtime = new Date().getTime();
                double latitude = (Math.random() * (2 * 85.05112878)) - 85.05112878;
                double longitude = (Math.random() * 360.0) - 180.0;
                String msg = runtime + "," + latitude + "," + longitude;
                try {
                    ProducerRecord<String, String> data = new
                            ProducerRecord<String, String>("ufo_sightings", key, msg);
                    producer.send(data);
                    long wait = Math.round(Math.random() * 25);
                    Thread.sleep(wait);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

SimpleConsumer.java

Note that this consumer is designed as an infinite loop. In normal operation of Kafka, all the producers could be idle while consumers are likely to be still running.

The example includes Java properties for setting up the client identified in the comments; the functional parts of the code are in **bold**. This code is compatible with versions as old as the 0.9.0-kafka-2.0.0 version of Kafka.

```
package com.cloudera.kafkaexamples;

import java.util.Arrays;
import java.util.Properties;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.serialization.StringDeserializer;

public class SimpleConsumer {
    public static void main(String[] args) {

        // Set up client Java properties
        Properties props = new Properties();
        props.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
                "host1:9092,host2:9092,host3:9092");
        // Just a user-defined string to identify the consumer group
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "test");
        // Enable auto offset commit
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
        props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "1000");
        props.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
                StringDeserializer.class.getName());
        props.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
                StringDeserializer.class.getName());

        try (KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props)) {
            // List of topics to subscribe to
            consumer.subscribe(Arrays.asList("ufo_sightings"));
            while (true) {
                try {
                    ConsumerRecords<String, String> records = consumer.poll(100);
                    for (ConsumerRecord<String, String> record : records) {
```

```
                        System.out.printf("Offset = %d\n", record.offset());
                        System.out.printf("Key    = %s\n", record.key());
                        System.out.printf("Value  = %s\n", record.value());
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Moving Kafka Clients to Production

Now that you've seen the basic examples of a producer and consumer, prototyping your own designs shouldn't be too difficult. However, your code will likely undergo several iterations that improve on scalability, debuggability, robustness, and maintainability.

This section presents recommendations in the form of code snippets that illustrate some of the important ways to use the producer and consumer APIs.

Reuse your Producer/Consumer object

In these examples, the consumer constructor should be called once and the `poll()` method called within a loop. If this object is not reused, then a new connection to the broker is opened with each new `KafkaConsumer` object created.

Recommended

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);

while (true) {
  ConsumerRecords<String, String> records = consumer.poll(100);
}
```

Not Recommended

```
while (true) {
  KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
  ConsumerRecords<String, String> records = consumer.poll(100);
}
```

Similarly, it is recommended that you use one `KafkaConsumer` and/or `KafkaProducer` object per thread. Creating more objects opens multiple ports per broker connection. Overusing ephemeral ports can cause performance issues.

In addition, Cloudera recommends to set and use a fixed `client.id` for producers and consumers when they are connecting to the brokers. If this is not done, Kafka will assign a new client id every time a new connection is established, which can severely increase resource utilization (memory) on the broker side.

Each KafkaConsumer object requires calling poll() frequently

As explained in the Apache Kafka documentation topic New Consumer Configs, any consumer connected to a partition will time out if `poll()` is not called within the period defined by `max.poll.interval.ms`.

In the example below, the call to `myDataProcess.doStuff(records)` can cause `poll()` to be called infrequently. This could be due to a combination of reasons:

- Being a blocking method call.
- Doing work on a remote machine.
- Having highly variable processing time.
- Saving to storage that has highly variable I/O throughput.

In such cases, consider having another thread or process doing the actual work and making the handoff as lightweight as possible.

Example: `poll()` gets KafkaException due to session timeout

```
while (true) {
  KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
  ConsumerRecords<String, String> records = consumer.poll(100);
  // the call below should return quickly in all cases
  myDataProcess.doStuff(records);
}
```

Catch all exceptions from poll()

From the `poll()` Javadoc page, you can see that the `poll()` method throws several exceptions. If the catch statements (**`bold`** in the example) are not complete, then any uncaught exception will end up in the finally statement calling `KafkaConsumer#close()`. This will not be the desired behavior in many cases.

```
while (true) {
  try {
    ConsumerRecords<String, String> records = consumer.poll(100);
  } catch (Exception e) {
    e.printStackTrace();
  } finally {
    consumer.close();
  }
```

Callback#onCompletion() should always exit without errors

The interface `org.apache.kafka.clients.producer.Callback` (Javadoc) is used to define a class that can be used upon completion of a `KafkaProducer#send()` call. It allows for tracking, clean up, or other administrative code to be called. An example of unintended usage is to call `KafkaProducer#send()` within the `Callback#onCompletion()` method, essentially mimicking a retry. Because the `onCompletion()` method is always expected to return cleanly and the `send()` method makes no such guarantees, calling `send()` within the callback could end up hanging the code in case of network or broker issues.

Check your API usage against the latest API

The documentation for the latest upstream release of Apache Kafka indicates if there have been any changes to how the APIs are used (`setup`, `read`, `write`). Reviewing the latest information could help avoid upgrade-related changes to your producer or consumer.

Some examples from past versions include:

Old Class or Package	New Class or Package
`kafka.producer.ProducerConfig`	`java.util.Properties`
`kafka.javaapi.*`	`kafka.api.*`
`kafka.producer.KeyedMessage`	`kafka.clients.producer.ProducerRecord`

Hidden Dependency on Network Availability

Network dependency is one of the more subtle issues. Given the consumer dependencies on Sentry and Zookeeper, having a combination of frequent or prolonged DNS or network outages can also cause various session timeouts to occur. Such timeouts will force partition rebalancing on the brokers, which will worsen general Kafka reliability.

Should these issues be common in your network, you may need to have a less straightforward design that can handle such reliability issues outside of the Kafka client.

Read the Details Carefully in the Apache Kafka Javadoc

The following pages have additional details about Kafka client programming:

- KafkaConsumer Javadoc

- KafkaProducer Javadoc

These Javadoc pages are quite dense with information. They assume you have sufficient background in reliable computing, networking, multithreading, and distributed systems to use the APIs correctly. While the previous sections point out many caveats in using the client APIs, the Javadoc (and ultimately the source code) provides a more detailed explanation.

Kafka Metrics

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 4: Metrics by Category

Category	Cloudera Manager Metrics Doc	Apache Kafka Metrics Doc
Cloudera Manager Kafka Service	Base Metrics on page 94	
Broker	Broker Metrics on page 95 Broker Topic Metrics on page 163 Replica Metrics on page 168	Broker
Common Producer/Consumer		Client Client-to-Broker
Producer		Producer Producer Sender
Consumer		Consumer Group Consumer Fetch
Mirror Maker	Mirror Maker Metrics on page 167	Same as Producer or Consumer tables

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:

- The JDK comes with the `jconsole` utility.
- VisualVM has a MBeans plugin.

Building Cloudera Manager Charts with Kafka Metrics

The Charts edit menu looks like a small pencil icon in the Charts page of the Cloudera Manager console. From there, choose **Add from Chart Builder** and enter a query for the appropriate metric.

```
SELECT
    metric
WHERE
    filter
```

Some specific examples of queries for Cloudera Metrics are:

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
```

Kafka Administration

This section describes managing a Kafka cluster in production, including:

Kafka Administration Basics

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.

There are many more variables available for fine-tuning broker log management. For more detailed information, look at the relevant variables in the Apache Kafka documentation topic Broker Configs.

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

Record Management

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

Both broker JVM garbage collection and JVM garbage log rotation is enabled by default in the Kafka version delivered with CDH. 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 with the **Kafka Broker Environment Advanced Configuration Snippet (Safety Valve)** property.

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.

Adding Users as Kafka Administrators

In some cases, additional users besides the `kafka` account need administrator access. This can be done in Cloudera Manager by going to **Kafka** > **Configuration** > **Super users**.

Migrating Brokers in a Cluster

Brokers can be moved to a new host in a Kafka cluster. This might be needed in the case of catastrophic hardware failure. Make sure the following are true before starting:

- Make sure the cluster is healthy.
- Make sure all replicas are in sync.
- Perform the migration when there is minimal load on the cluster.

Brokers need to be moved one-by-one. There are two techniques available:

Using `kafka-reassign-partitions` tool

This method involves more manual work to modify JSON, but does not require manual edits to configuration files. For more information, see kafka-reassign-partitions on page 66.

Modify the broker IDs in `meta.properties`

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

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.

   See Using rsync to Copy Files from One Broker to Another on page 62.

5. Start up the new broker.

   It re-replicates data from the other nodes.

Note that 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.

To avoid performance degradation of business workloads, you can limit the resources that these background processes can consume by specifying the `-throttle`parameter when running `kafka-reassign-partitions`.

Using rsync to Copy Files from One Broker to Another

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

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`.

Setting User Limits for Kafka

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.

See Viewing Charts for Cluster, Service, Role, and Host Instances.

Quotas

For a quick video introduction to quotas, see Quotas.

In CDK 2.0 Powered by Apache Kafka and higher, 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.

Setting Quotas

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**. For more information, see Modifying Configuration Properties Using Cloudera Manager.

Kafka Administration Using Command Line Tools

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 more easily and conveniently using Cloudera Manager. Therefore, before you continue, make sure to review Unsupported Command Line Tools on page 63 and Notes on Kafka CLI Administration on page 64.

Note: Output examples in this document are cleaned and formatted for easier readability.

Unsupported Command Line Tools

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-distributed` `connect-standalone`	Kafka Connect is currently not supported.
`kafka-acls`	Cloudera recommends using Sentry to manage ACLs instead of this tool.
`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`	Do not use with CDH.
`kafka-mirror-maker`	Use Cloudera Manager to create any CDH Mirror Maker instance.
`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. It is recommended to use `kafka-reassign-partitions` instead of `kafka-preferred-replica-election`.
`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-simple-consumer-shell`	Deprecated in Apache Kafka.
`kafka-streams-application-reset`	Kafka Streams is currently not supported.
`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.

Notes on Kafka CLI Administration

Here are some additional points to be aware of regarding Kafka administration:

- 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.

kafka-topics

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

```
kafka-topics --zookeeper zkhost --describe
```

```
Topic: topic-a1      PartitionCount:3      ReplicationFactor:3      Configs:
         Topic: topic-a1      Partition: 0      Leader: 64      Replicas: 64,62,63
  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: 63,64,62
  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

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=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
```

The Apache Kafka documentation includes a complete list of topic properties.

kafka-console-consumer

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>... --topic <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-console-producer

This tool is used to write messages to a topic. It is typically not as useful as the console consumer, but it can be useful when the messages are in a text based format. In general, the usage will be something like:

```
cat file | kafka-console-producer args
```

kafka-consumer-groups

The basic usage of the `kafka-consumer-groups` tool is:

```
kafka-consumer-groups --bootstrap-server broker1,broker2... --describe --group GROUP_ID
```

This tool is primarily useful for debugging consumer offset issues. The output from the tool shows the log and consumer offsets for each partition connected to the consumer group corresponding to `GROUP_ID`. You can see at a glance which consumers are current with their partition and which ones are behind. From there, you can determine which partitions (and likely the corresponding brokers) are slow.

Beyond this debugging usage, there are other more advanced options to this tool:

- `--execute --reset-offsets` *`SCENARIO_OPTION`*: Resets the offsets for a consumer group to a particular value based on the SCENARIO_OPTION flag given.

  Valid flags for *SCENARIO_OPTION* are:

  - `--to-datetime`
  - `--by-period`
  - `--to-earliest`
  - `--to-latest`
  - `--shift-by`
  - `--from-file`
  - `--to-current`

  You will likely want to set the `--topic` flag to restrict this change to a specific topic or a specific set of partitions within that topic.

This tool can be used to reset all offsets on all topics. This is something you probably won't ever want to do. It is highly recommended that you use this command carefully.

kafka-reassign-partitions

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 on page 67
- Reassignment Configuration JSON on page 67

Topics-to-Move JSON

This JSON file specifies the topics that you want to reassign. This 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-reasssign-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 5: 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

Property	Description
	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

To reassign partitions, complete the following steps:

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.

   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. 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. 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
```

Examples

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. Changes to the original example are marked in bold.

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"]}]}
```

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

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.

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 Contents of the kafka-log-dirs Output table gives an overview of the information provided by the `kafka-log-dirs` tool.

Table 6: Contents of the kafka-log-dirs Output

Property	Description
`broker`	Displays the ID of the broker.

Property	Description
`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.

Tool Usage

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"
```

zookeeper-security-migration

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 Kafka Security Hardening with Zookeeper ACLs on page 83.

Tool Usage

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
```

kafka-delegation-tokens

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

Tool Usage

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 used to authenticate, the brokers will remove expired delegation tokens from the broker's cache and from Zookeeper.

```
kafka-delegation-tokens --bootstrap-server hostname:port --renew    --renew-time-period
 -1 --command-config client.properties --hmac lAYYSFmLs4bTjf+lTZ1LCHR/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+lTZ1LCHR/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. At the moment, this functionality is not available in the Kafka-Sentry binding.

kafka-*-perf-test

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

- Measuring read and/or write throughput.
- Stress testing the cluster based on specific parameters (such as message size).
- Load testing 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.

Enabling DEBUG or TRACE in command line scripts

In some cases, you may find it useful to produce extra debugging output from the Kafka client API. The DEBUG (shown below) or TRACE levels can be set by replacing the setting in the log4j properties file as follows:

```
cp /etc/kafka/conf/tools-log4j.properties /var/tmp
sed -i -e 's/WARN/DEBUG/g' /var/tmp/tools-log4j.properties

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

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.

Disk Management

Monitoring

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. Alters are raised when failures are detected. For more information, see Cloudera Manager Health Tests.

Disk Capacity

Monitor free space on mounted disks and open file descriptors. For more information, see Useful Shell Command Reference on page 169. Reassign partitions or move log files around if necessary. For more information, see kafka-reassign-partitions on page 66.

Handling 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 on page 70 tool, or by reviewing the error messages of `KafkaStorageException` entries in the Kafka broker log file.

To view the Kafka broker log file, complete the following steps:

1. In Cloudera Manager go to the Kafka service, select **Instances** and select the broker.
2. Go to **Log Files** > **Role 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.

Disk Replacement

To replace a disk, complete the following steps:

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 **Actions** > **Gracefully 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, select**Instances** and select the broker.
   b. Go to **Actions** > **Start this Kafka Broker**.

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

Disk Removal

To remove a disk from the configuration, complete the following steps:

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 **Actions** > **Gracefully 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 **Actions** > **Start this Kafka Broker**.

   The Kafka broker redistributes data across the cluster.

Reassigning Replicas 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 documentation for the kafka-reassign-partitions on page 66 tool.

Retrieving Log Directory Replica Assignment Information

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 documentation for the kafka-log-dirs on page 70 tool.

JBOD

As of CDH 6.1.0, Kafka clusters with nodes using JBOD configurations are supported by Cloudera.

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 and Migration

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, see Upgrading the CDH Cluster.

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

Setup

To set up JBOD in your Kafka environment, perform the following steps:

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

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

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

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

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

Migration

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. For a full tool description, see kafka-reassign-partitions on page 66.

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.

Prerequisites

- Set up JBOD in your Kafka environment. For more information, see Setup on page 76.
- 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.

Steps

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

To migrate data to JBOD configured disks, perform the following steps:

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.

   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. 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"]}
```

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

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

5. Save the file.

6. Start the redistribution process with the following command:

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

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

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

```
Current partition replica assignment

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

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

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

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

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

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

Kafka Delegation Tokens

Delegation tokens were introduced as a lightweight authentication method to complement existing SASL authentication. Kafka is designed to support a high number of clients. However, using Kerberos authentication might be difficult in some environments due to the following reasons:

- With Kerberos authentication, all clients need access to a keytab or a TGT. Securely distributing the keytabs requires a lot of effort and careful administration. When the TGT is compromised, it has a high blast radius, especially when the same keytabs are used to access multiple services.
- With Kerberos, client authentication is centralized, and the high number of clients can put a high load on the KDC (Key Distribution Center), resulting in a bottleneck.

Many Hadoop components use delegation tokens to mitigate these problems:

- Delegation tokens allow these components to secure distributed workloads with low administrative overhead
- It is not required to distribute a Kerberos TGT or keytab, which, if compromised, may grant access to all services.
- A Delegation token is strictly tied to its associated service causing less damage if exposed.
- Delegation tokens make credential renewal more lightweight. This is because the renewal is designed in such a way that only the renewer and the service are involved in the renewal process. The token itself remains the same, so parties already using the token do not have to be updated.

Delegation Token Basics

Kafka delegation tokens were modeled after Hadoop delegation tokens, and many of their mechanism are the same or very similar. However, this does not mean that they are interchangeable.

Delegation tokens are generated and verified following the HMAC mechanism. There are two basic parts of information in a delegation Token:

- Public part (tokenID)
- Private part (HMAC value)

The following steps describe the basics of how delegation tokens are used:

1. The user initially authenticates with the Kafka cluster via SASL, and obtains a delegation token using either the `AdminClient` APIs or the `kafka-delegation-token` tool. The principal that created the delegation token is its owner.
2. The delegation token details are securely passed to Kafka clients. This can be achieved by sending the token data over an SSL/TLS encrypted connection or writing them to a secure shared storage.
3. Instead of using Kerberos, the Kafka client uses the delegation tokens for subsequent authentication with the brokers. See The Client authentication using delegation tokens.
4. The token is valid for a certain time period, but it can be:

   Renewed

   A delegation token can be renewed multiple times up until its maximum life before it expires. The token can be renewed by the owner or any other principals the owner sets as “renewer” at time of creation.

   Revoked

   A delegation token can be revoked ahead of its expiry time.

Broker Configuration Settings

Certain delegation token properties can be configured on a service level in Cloudera Manager. Table 7: Delegation Token Broker Configuration Settings on page 79gives an overview of these properties.

Table 7: Delegation Token Broker Configuration Settings

Name	Description	Type	Default
`delegation.token.enable`	Enables authentication with delegation tokens for this Kafka service. When enabled, a secure password is automatically generated and used as the "delegation.token.master.key" property for Kafka Brokers. Only	boolean	false

	allowed if Kerberos authentication is enabled.		
delegation.token.expiry.time.ms	The expiry time of a delegation token. A delegation token can be renewed before its expiry time to extend its validity up to its maximum lifetime. If it is not renewed, it will expire even if it has time remaining from its maximum lifetime.	time	1 day
delegation.token.max.lifetime.ms	The maximum amount of time that a delegation token can be valid for.	time	7 days

Enable Authentication with Delegation Tokens

Although the following steps enable authentication between clients and servers using the SASL/SCRAM mechanism, it is only as a vehicle for delegation tokens. Using SCRAM credentials is not supported otherwise. Sensitive delegation token metadata is stored in Zookeeper. It is recommended to restrict access on Zookeeper nodes to prevent access to sensitive delegation token related data through Zookeeper. As the connection between Kafka and Zookeeper is not encrypted, it is also recommended to use delegation tokens only if no unauthorized person can read and manipulate the traffic between these services.

For more information on restricting Zookeeper access, see Kafka Security Hardening with Zookeeper ACLs on page 83.

Prerequisites

A secure Kafka cluster with Kerberos authentication enabled is required. For more information, see Enabling Kerberos Authentication on page 35.

Steps

Enable Authentication with delegation tokens by completing the following steps:

1. In Cloudera Manager go to the Kafka service.
2. Select **Configuration** and find the **Enable Delegation Tokens** property.
3. Enable delegation tokens for all required services by checking the checkbox next to the name of the service.
4. Click **Save Changes**.
5. Perform a **Rolling Restart**.
    a. Return to the Home page by clicking the Cloudera Manager logo.
    b. Go to the Kafka service and select **Actions** > **Rolling Restart**.
    c. Check the **Restart roles with stale configurations only** checkbox and click **Rolling restart**.
    d. Click **Close** when the restart has finished.

Completing these steps generates the necessary secrets and settings for delegation tokens.

Managing Individual Delegation Tokens

The functionality that's needed to manage and use delegation tokens is accessible using the AdminClient APIs or the `kafka-delegation-tokens` tool. All of their operations are allowed only via SASL authenticated channels.

Both the API and the script provide the following actions:

> **Note:** The examples presented show how these actions can be executed with the `kafka-delegation-tokens` tool.

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 used to authenticate, the brokers will remove expired delegation tokens from the broker's cache and from Zookeeper.

```
kafka-delegation-tokens --bootstrap-server hostname:port --renew    --renew-time-period
 -1 --command-config client.properties --hmac lAYYSFmLs4bTjf+lTZ1LCHR/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+lTZ1LCHR/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. At the moment, this functionality is not available in the Kafka-Sentry binding.

Rotating the Master Key/Secret

The brokers generate and verify delegation tokens using a secret called `delegation.token.master.key`. This secret is generated by Cloudera Manager and securely passed to Kafka brokers when authentication with delegation tokens is enabled. You can change the secret with the Cloudera Manager API. This should be done if the secret becomes compromised, or simply as a precautionary measure.

To change the secret, complete these steps:

> **Important:** Clients that were already connected to brokers before starting the process, will continue to work even after the master key/secret is rotated. However, any new connections (authentication requests), as well as renew and expire requests with old tokens can fail.

1. Expire existing tokens.

`kafka-delegation-tokens` example command:

```
kafka-delegation-tokens --bootstrap-server hostname:port --expire --expiry-time-period
 -1 --command-config client.properties --hmac lAYYSFmLs4bTjf+lTZ1LCHR/ZZFNA==
```

2. Generate a new master key

```
curl -X PUT -u "user" -H "content-type:application/json" -i "https://cloudera manager
host:7183/api/v31/clusters/cluster name/services/kafka service name/config" -d '{"items"
 : [ {"name" : "delegation.token.master.key","value" : "'$(openssl rand -base64
24)'","sensitive" : true}]}'
```

3. Perform a Rolling Restart
   a. In Cloudera Manager go to the Kafka service and select **Actions** > **Rolling Restart**.
   b. Check the **Restart roles with stale configurations only** checkbox and click Rolling restart.
   c. Click **Close** when the restart has finished.
4. Reauthenticate with all clients. This will generate the new tokens.

Client Authentication using Delegation Tokens

Brokers authenticate clients by verifying the delegation tokens provided by the client against the stored delegation tokens. Delegation token authentication makes use of SASL/SCRAM authentication mechanism under the hood. You can configure Kafka clients in two ways, to use individually assigned delegation tokens or to use a common delegation token.

Configuring Clients on a Producer or Consumer Level

You can set up client authentication by configuring the JAAS configuration property for each client. The JAAS configuration property can be set in the `producer.properties` or `consumer.properties` file of the client. With this configuration method, you have the ability to specify different token details for each Kafka client within a JVM. As a result you can configure Kafka clients in a way that each of them use a unique token for authentication.

Example Configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule  required \
    username="tokenID" \
    password="lAYYSFmLs4bTjf+lTZ1LCHR/ZZFNA==" \
    tokenauth="true";
```

There are three options that need to be specified. These are the `username`, `password` and `tokenauth` options.

The `username` and `password` options specify the token ID and token HMAC. The `tokenauth` option expresses the intent to use token authentication to the server.

Configure Clients on an Application Level

With this configuration method, you can set up all clients within a JVM to use the same delegation token for authentication.

Configure Clients to use a common delegation token by completing these steps:

1. Add a KafkaClient entry with a login module item to your JAAS configuration file. The module has to specify the `username`, `password` and `tokenauth` options.

   Example:

```
KafkaClient {
org.apache.kafka.common.security.scram.ScramLoginModule  required
    username="tokenID"
    password="lAYYSFmLs4bTjf+lTZ1LCHR/ZZFNA=="
    tokenauth="true";
}
```

2. Pass the location of your JAAS configuration file as a JVM parameter through a command line interface. This will set the JAAS configuration on the Java process level.

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

Kafka Security Hardening with Zookeeper ACLs

Restricting Access to Kafka Metadata in Zookeeper

Locking down znodes in Zookeeper can be used to protect Kafka metadata against unauthorized access. Direct manipulation of metadata in Zookeeper is not only dangerous for the health of the cluster, but can also serve as an entry point for malicious users to gain elevated access who can then alter the owner or renewer of delegation tokens.

Prerequisites

A secure Kafka cluster with Kerberos authentication enabled is required. For more information see, Enabling Kerberos Authentication on page 35.

Steps

Restrict access to Kafka metadata stored in ZooKeeper by completing the following steps:

1. Enable the use of secure ACLs by setting `zookeeper.set.acl` configuration parameter to true.
   a. In Cloudera Manager go to the Kafka service.
   b. Select **Configuration** and find the **Kafka Broker Advanced Configuration Snippet (Safety Valve) for kafka.properties** property.
   c. Add the following line to the **Kafka Broker Advanced Configuration Snippet (Safety Valve) for kafka.properties** property:

```
zookeeper.set.acl=true
```

2. Perform a Rolling Restart
   a. Return to the Home page by clicking the Cloudera Manager logo.
   b. Go to the Kafka service and select **Actions Rolling Restart**.
   c. Check the **Restart roles with stale configurations only** checkbox and click **Rolling restart**.
   d. Click **Close** when the restart has finished.
3. Pass the JAAS config file location as a JVM parameter through a command line interface. You can do this by setting the value of the KAFKA_OPTS environment variable to `-Djava.security.auth.login.config=path/to/jaas.conf`.

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

4. Run the `zookeeper-security-migration` tool with the `zookeeper.acl` option set to `secure`.

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

   The tool traverses the corresponding sub-trees changing the ACLs of the znodes

5. Reset the ACLs on the root node to allow full access.

   Resetting the ACLS on the root node is required because the `zookeeper-security-migration` tool also changes the ACLs on the root znode. This leads to the failure of the Zookeeper canary tests, which subsequently makes the service display as unhealthy in Cloudera Manager.

> **Important:** This step is only necessary if the `zookeeper.chroot` parameter of the broker is set to `/`.

> **Note:** Because the Kafka metadata at this point is already restricted, only authorized users or Zookeeper super users can complete this step.

a. Change the JVMFLAGS environment variable to `-Djava.security.auth.login.config=*path/to/jaas.conf*`

```
export JVMFLAGS="-Djava.security.auth.login.config=path/to/jaas.conf"
```

b. Start the zookeeper client

```
zookeeper-client -server $(hostname -f):2181
```

c. Enter the following to reset the ACLs of the root node:

```
setAcl / world:anyone:crdwa
```

Once Kafka metadata in Zookeeper is restricted via ACLS, administrative operations, for example topic creation, deletion, any configuration changes and so on, can only be performed by authorized users.

Unlocking Kafka Metadata in Zookeeper

Prerequisites

A secure Kafka cluster with Kerberos authentication enabled is required. For more information see, Enabling Kerberos Authentication on page 35.

Steps

In order to unrestrict access to Kafka metadata stored in Zookeeper by completing the following steps:

1. Disable the use of secure ACLs by setting `zookeeper.set.acl` configuration parameter to `false`.
   a. In Cloudera Manager go to the Kafka service.
   b. Select **Configuration** and find the **Kafka Broker Advanced Configuration Snippet (Safety Valve) for kafka.properties** property.
   c. Add the following line to the **Kafka Broker Advanced Configuration Snippet (Safety Valve) for kafka.properties** property:

```
zookeeper.set.acl=false
```

2. Perform a Rolling Restart
   a. Return to the Home page by clicking the Cloudera Manager logo.
   b. Go to the Kafka service and select **Actions Rolling Restart**.
   c. Check the **Restart roles with stale configurations only** checkbox and click **Rolling restart**.
   d. Click **Close** when the restart has finished.
3. Run the `zookeeper-security-migration` tool with the `zookeeper.acl` option set to `unsecure`.

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

The tool traverses the corresponding sub-trees changing the ACLs of the znodes.

Kafka Performance Tuning

Performance tuning involves two important metrics:

- Latency measures how long it takes to process one event.
- Throughput measures how many events arrive within a specific amount of time.

Most systems are optimized for either latency or throughput. Kafka is balanced for both. A well-tuned Kafka system has just enough brokers to handle topic throughput, given the latency required to process information as it is received.

Tuning your producers, brokers, and consumers to send, process, and receive the largest possible batches within a manageable amount of time results in the best balance of latency and throughput for your Kafka cluster.

The following sections introduce the concepts you'll need to be able to balance your Kafka workload and then provide practical tuning configuration to address specific circumstances.

For a quick video introduction to tuning Kafka, see Tuning Your Apache Kafka Cluster.

There are a few concepts described here that will help you focus your tuning efforts. Additional topics in this section provide practical tuning guidelines:

Tuning Brokers

Topics are divided into partitions. Each partition has a leader. Topics that are properly configured for reliability will consist of a leader partition and 2 or more follower partitions. When the leaders are not balanced properly, one might be overworked, compared to others.

Depending on your system and how critical your data is, you want to be sure that you have sufficient replication sets to preserve your data. For each topic, Cloudera recommends starting with one partition per physical storage disk and one consumer per partition.

Tuning Producers

Kafka uses an asynchronous publish/subscribe model. When your producer calls `send()`, the result returned is a future. The future provides methods to let you check the status of the information in process. When the batch is ready, the producer sends it to the broker. The Kafka broker waits for an event, receives the result, and then responds that the transaction is complete.

If you do not use a future, you could get just one record, wait for the result, and then send a response. Latency is very low, but so is throughput. If each transaction takes 5 ms, throughput is 200 events per second — slower than the expected 100,000 events per second.

When you use `Producer.send()`, you fill up buffers on the producer. When a buffer is full, the producer sends the buffer to the Kafka broker and begins to refill the buffer.

Two parameters are particularly important for latency and throughput: `batch size` and `linger time`.

Batch Size

`batch.size` measures batch size in total bytes instead of the number of messages. It controls how many bytes of data to collect before sending messages to the Kafka broker. Set this as high as possible, without exceeding available memory. The default value is 16384.

If you increase the size of your buffer, it might never get full. The Producer sends the information eventually, based on other triggers, such as linger time in milliseconds. Although you can impair memory usage by setting the buffer batch size too high, this does not impact latency.

If your producer is sending all the time, you are probably getting the best throughput possible. If the producer is often idle, you might not be writing enough data to warrant the current allocation of resources.

Linger Time

`linger.ms` sets the maximum time to buffer data in asynchronous mode. For example, the setting of 100 means that it batches 100ms of messages to send at once. This improves throughput, but the buffering adds message delivery latency.

By default, the producer does not wait. It sends the buffer any time data is available.

Instead of sending immediately, you can set `linger.ms` to 5 and send more messages in one batch. This would reduce the number of requests sent, but would add up to 5 milliseconds of latency to records sent, even if the load on the system does not warrant the delay.

The farther away the broker is from the producer, the more overhead required to send messages. Increase `linger.ms` for higher latency and higher throughput in your producer.

Tuning Consumers

Consumers can create throughput issues on the other side of the pipeline. The maximum number of consumers in a consumer group for a topic is equal to the number of partitions. You need enough partitions to handle all the consumers needed to keep up with the producers.

Consumers in the same consumer group split the partitions among them. Adding more consumers to a group can enhance performance (up to the number of partitions). Adding more consumer groups does not affect performance.

Mirror Maker Performance

Kafka Mirror Maker is a tool to replicate topics between data centers. It is best to run Mirror Maker at the destination data center. Consuming messages from a distant cluster and writing them into a local cluster tends to be more safe than producing over a long-distance network. Deploying the Mirror Maker in the source data center and producing remotely has a higher risk of losing data. However, if you need this setup, make sure that you configure `acks=all` with appropriate number of retries and min ISR.

- Encrypting data in transit with SSL has impact on performance of Kafka brokers.
- To reduce lag between clusters, you can improve performance by deploying multiple Mirror Maker instances using the same consumer group ID.
- Measure CPU utilization.
- Consider using compression for consumers and producers when mirroring topics between data centers as bandwidth can be a bottleneck.
- Monitor lag and metrics of Mirror Maker.

To properly size Mirror Maker, take expected throughput and maximum allowed lag between data centers into account.

- `num.streams` parameter controls the number of consumer threads in Mirror Maker.
- `kafka-producer-perf-test` can be used to generate load on the source cluster. You can test and measure performance of Mirror Maker with different `num.streams` values (start from 1 and increase it gradually).

Good performance can be achieved with proper consumer and producer settings and properly tuned OS properties, such as networking and I/O related kernel settings.

Kafka Tuning: Handling Large Messages

Before configuring Kafka to handle large messages, first consider the following options to reduce message size:

- The Kafka producer can compress messages. For example, if the original message is a text-based format (such as XML), in most cases the compressed message will be sufficiently small.
- Use the `compression.type` producer configuration parameters to enable compression. gzip, lz4 and Snappy are supported.
- If shared storage (such as NAS, HDFS, or S3) is available, consider placing large files on the shared storage and using Kafka to send a message with the file location. In many cases, this can be much faster than using Kafka to send the large file itself.
- Split large messages into 1 KB segments with the producing client, using partition keys to ensure that all segments are sent to the same Kafka partition in the correct order. The consuming client can then reconstruct the original large message.

If you still need to send large messages with Kafka, modify the configuration parameters presented in the following sections to match your requirements.

Table 8: Broker Configuration Properties

Property	Default Value	Description
`message.max.bytes`	1000000 (1 MB)	Maximum message size the broker accepts.
`log.segment.bytes`	1073741824 (1 GiB)	Size of a Kafka data file. Must be larger than any single message.
`replica.fetch.max.bytes`	1048576 (1 MiB)	Maximum message size a broker can replicate. Must be larger than `message.max.bytes`, or a broker can accept messages it cannot replicate, potentially resulting in data loss.

Table 9: Consumer Configuration Properties

Property	Default Value	Description
`max.partition.fetch.bytes`	1048576 (10 MiB)	The maximum amount of data per-partition the server will return.
`fetch.max.bytes`	52428800 (50 MiB)	The maximum amount of data the server should return for a fetch request.

Note: The consumer is able to consume a message batch that is larger than the default value of the `max.partition.fetch.bytes` or `fetch.max.bytes` property. However, the batch will be sent alone, which can cause performance degradation.

Kafka Cluster Sizing

Cluster Sizing - Network and Disk Message Throughput

There are many variables that go into determining the correct hardware footprint for a Kafka cluster. The most accurate way to model your use case is to simulate the load you expect on your own hardware. You can do this using the load generation tools that ship with Kafka, `kafka-producer-perf-test` and `kafka-consumer-perf-test`. For more information, see Kafka Administration Using Command Line Tools on page 63.

However, if you want to size a cluster without simulation, a very simple rule could be to size the cluster based on the amount of disk-space required (which can be computed from the estimated rate at which you get data times the required data retention period).

A slightly more sophisticated estimation can be done based on network and disk throughput requirements. To make this estimation, let's plan for a use case with the following characteristics:

- `W` - MB/sec of data that will be written
- `R` - Replication factor
- `C` - Number of consumer groups, that is the number of readers for each write

Kafka is mostly limited by the disk and network throughput.

The volume of writing expected is `W * R` (that is, each replica writes each message). Data is read by replicas as part of the internal cluster replication and also by consumers. Because every replicas but the master read each write, the read volume of replication is `(R-1) * W`. In addition each of the `C` consumers reads each write, so there will be a read volume of `C * W`. This gives the following:

- Writes: `W * R`
- Reads: `(R+C- 1) * W`

However, note that reads may actually be cached, in which case no actual disk I/O happens. We can model the effect of caching fairly easily. If the cluster has `M` MB of memory, then a write rate of `W` MB/second allows `M/(W * R)` seconds of writes to be cached. So a server with 32 GB of memory taking writes at 50 MB/second serves roughly the last 10 minutes of data from cache. Readers may fall out of cache for a variety of reasons—a slow consumer or a failed server that recovers and needs to catch up. An easy way to model this is to assume a number of lagging readers you to budget for. To model this, let's call the number of lagging readers `L`. A very pessimistic assumption would be that `L = R + C -1`, that is that all consumers are lagging all the time. A more realistic assumption might be to assume no more than two consumers are lagging at any given time.

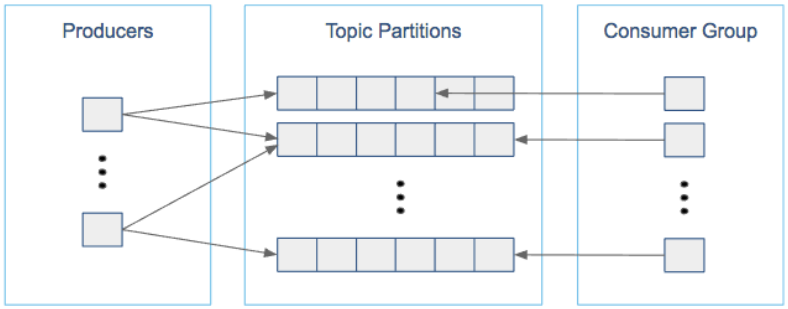
Based on this, we can calculate our cluster-wide I/O requirements:

- Disk Throughput (Read + Write): `W * R + L * W`
- Network Read Throughput: `(R + C -1) * W`
- Network Write Throughput: `W * R`

A single server provides a given disk throughput as well as network throughput. For example, if you have a 1 Gigabit Ethernet card with full duplex, then that would give 125 MB/sec read and 125 MB/sec write; likewise 6 7200 SATA drives might give roughly 300 MB/sec read + write throughput. Once we know the total requirements, as well as what is provided by one machine, you can divide to get the total number of machines needed. This gives a machine count running at maximum capacity, assuming no overhead for network protocols, as well as perfect balance of data and load. Since there is protocol overhead as well as imbalance, you want to have at least 2x this ideal capacity to ensure sufficient capacity.

Choosing the Number of Partitions for a Topic

Choosing the proper number of partitions for a topic is the key to achieving a high degree of parallelism with respect to writes to and reads and to distribute load. Evenly distributed load over partitions is a key factor to have good throughput (avoid hot spots). Making a good decision requires estimation based on the desired throughput of producers and consumers per partition.



For example, if you want to be able to read 1 GB/sec, but your consumer is only able process 50 MB/sec, then you need at least 20 partitions and 20 consumers in the consumer group. Similarly, if you want to achieve the same for producers, and 1 producer can only write at 100 MB/sec, you need 10 partitions. In this case, if you have 20 partitions, you can maintain 1 GB/sec for producing and consuming messages. You should adjust the exact number of partitions to number of consumers or producers, so that each consumer and producer achieve their target throughput.

So a simple formula could be:

$$\#Partitions = max(N_P, N_C)$$

where:

- N_P is the number of required producers determined by calculating: T_T/T_P
- N_C is the number of required consumers determined by calculating: T_T/T_C
- T_T is the total expected throughput for our system
- T_P is the max throughput of a single producer to a single partition
- T_C is the max throughput of a single consumer from a single partition

This calculation gives you a rough indication of the number of partitions. It's a good place to start. Keep in mind the following considerations for improving the number of partitions after you have your system in place:

- The number of partitions can be specified at topic creation time or later.
- Increasing the number of partitions also affects the number of open file descriptors. So make sure you set file descriptor limit properly.
- Reassigning partitions can be very expensive, and therefore it's better to over- than under-provision.
- Changing the number of partitions that are based on keys is challenging and involves manual copying (see Kafka Administration on page 60).
- Reducing the number of partitions is not currently supported. Instead, create a new a topic with a lower number of partitions and copy over existing data.
- Metadata about partitions are stored in ZooKeeper in the form of `znodes`. Having a large number of partitions has effects on ZooKeeper and on client resources:
  - Unneeded partitions put extra pressure on ZooKeeper (more network requests), and might introduce delay in controller and/or partition leader election if a broker goes down.
  - Producer and consumer clients need more memory, because they need to keep track of more partitions and also buffer data for all partitions.
- As guideline for optimal performance, you should not have more than 4000 partitions per broker and not more than 200,000 partitions in a cluster.

Make sure consumers don’t lag behind producers by monitoring consumer lag. To check consumers' position in a consumer group (that is, how far behind the end of the log they are), use the following command:

```
$ kafka-consumer-groups --bootstrap-server BROKER_ADDRESS --describe --group
CONSUMER_GROUP --new-consumer
```

Kafka Performance Broker Configuration

JVM and Garbage Collection

Garbage collection has a huge impact on performance of JVM based applications. It is recommended to use the Garbage-First (G1) garbage collector for Kafka broker. In Cloudera Manager specify the following under Additional Broker Java Options in the Kafka service configuration:

```
-server -XX:+UseG1GC -XX:MaxGCPauseMillis=20
-XX:InitiatingHeapOccupancyPercent=35 -XX:+DisableExplicitGC
-Djava.awt.headless=true -Djava.net.preferIPv4Stack=true
```

Cloudera recommends to set 4-8 GB of JVM heap size memory for the brokers depending on your use case. As Kafka's performance depends heavily on the operating systems page cache, it is not recommended to collocate with other memory-hungry applications.

- Large messages can cause longer garbage collection (GC) pauses as brokers allocate large chunks. Monitor the GC log and the server log.

  Add this to Broker Java Options:

```
 -XX:+PrintGC -XX:+PrintGCDetails
 -XX:+PrintGCTimeStamps
-Xloggc:</path/to/file.txt>
```

- If long GC pauses cause Kafka to abandon the ZooKeeper session, you may need to configure longer timeout values, see Kafka-ZooKeeper Performance Tuning on page 93 for details.

Network and I/O Threads

Kafka brokers use network threads to handle client requests. Incoming requests (such as produce and fetch requests) are placed into a requests queue from where I/O threads are taking them up and process them. After a request is processed, the response is placed into an internal response queue from where a network thread picks it up and sends response back to the client.

- `num.network.threads` is an important cluster-wide setting that determines the number of threads used for handling network requests (that is, receiving requests and sending responses). Set this value mainly based on number of producers, consumers and replica fetchers.
- `queued.max.requests` controls how many requests are allowed in the request queue before blocking network threads.
- `num.io.threads` specifies the number of threads that a broker uses for processing requests from the request queue (might include disk I/O).

ISR Management

An in-sync replica (ISR) set for a topic partition contains all follower replicas that are caught-up with the leader partition, and are situated on a broker that is alive.

- If a replica lags "too far" behind from the partition leader, it is removed from the ISR set. The definition of what is too far is controlled by the configuration setting `replica.lag.time.max.ms`. If a follower hasn't sent any fetch requests or hasn't consumed up to the leaders log end offset for at least this time, the leader removes the follower from the ISR set.
- `num.replica.fetchers` is a cluster-wide configuration setting that controls how many fetcher threads are in a broker. These threads are responsible for replicating messages from a source broker (that is, where partition leader resides). Increasing this value results in higher I/O parallelism and fetcher throughput. Of course, there is a trade-off: brokers use more CPU and network.

- `replica.fetch.min.bytes` controls the minimum number of bytes to fetch from a follower replica. If there is not enough bytes, wait up to `replica.fetch.wait.max.ms`.
- `replica.fetch.wait.max.ms` controls how long to sleep before checking for new messages from a fetcher replica. This value should be less than `replica.lag.time.max.ms`, otherwise the replica is kicked out of the ISR set.
- To check the ISR set for topic partitions, run the following command:

```
kafka-topics --zookeeper ${ZOOKEEPER_HOSTNAME}:2181/kafka --describe --topic ${TOPIC}
```

- If a partition leader dies, a new leader is selected from the ISR set. There will be no data loss. If there is no ISR, unclean leader election can be used with the risk of data-loss.
- Unclean leader election occurs if `unclean.leader.election.enable` is set to true. By default, this is set to `false`.

Log Cleaner

As discussed in Record Management on page 60, the log cleaner implements log compaction. The following cluster-wide configuration settings can be used to fine tune log compaction:

- `log.cleaner.threads` controls how many background threads are responsible for log compaction. Increasing this value improves performance of log compaction at the cost of increased I/O activity.
- `log.cleaner.io.max.bytes.per.second` throttles log cleaner's I/O activity so that the sum of its read and write is less than this value on average.
- `log.cleaner.dedupe.buffer.size` specifies memory used for log compaction across all cleaner threads.
- `log.cleaner.io.buffer.size` controls total memory used for log cleaner I/O buffers across all cleaner threads.
- `log.cleaner.min.compaction.lag.ms` controls how long messages are left uncompacted.
- `log.cleaner.io.buffer.load.factor` controls log cleaner's load factor for the dedupe buffer. Increasing this value allows the system to clean more logs at once but increases hash collisions.
- `log.cleaner.backoff.ms` controls how long to wait until the next check if there is no log to compact.

Kafka Performance: System-Level Broker Tuning

Operating system related kernel parameters affect overall performance of Kafka. These parameters can be configured via `sysctl` at runtime. To make kernel configuration changes persistent (that is, use adjusted parameters after a reboot), edit `/etc/sysctl.conf`. The following sections describe some important kernel settings.

File Descriptor Limits

As Kafka works with many log segment files and network connections, the `Maximum Process File Descriptors` setting may need to be increased in some cases in production deployments, if a broker hosts many partitions. For example, a Kafka broker needs at least the following number of file descriptors to just track log segment files:

```
(number of partitions)*(partition size / segment size)
```

The broker needs additional file descriptors to communicate via network sockets with external parties (such as clients, other brokers, Zookeeper, Sentry, and Kerberos).

The `Maximum Process File Descriptors` setting can be monitored in Cloudera Manager and increased if usage requires a larger value than the default `ulimit` (often 64K). It should be reviewed for use case suitability.

- To review FD limit currently set for a running Kafka broker, run `cat /proc/KAFKA_BROKER_PID/limits`, and look for `Max open files`.
- To see open file descriptors, run:

```
lsof -p KAFKA_BROKER_PID
```

Filesystems

Linux records when a file was created (`ctime`), modified (`mtime`) and accessed (`atime`). The value `noatime` is a special mount option for filesystems (such as EXT4) in Linux that tells the kernel not to update inode information every time a file is accessed (that is, when it was last read). Using this option may result in write performance gain. Kafka is not relying on `atime`. The value `relatime` is another mounting option that optimizes how `atime` is persisted. Access time is only updated if the previous `atime` was earlier than the current modified time.

To view mounting options, run `mount -l` or `cat /etc/fstab` command.

Virtual Memory Handling

Kafka uses system page cache extensively for producing and consuming the messages. The Linux kernel parameter, `vm.swappiness`, is a value from 0-100 that controls the swapping of application data (as anonymous pages) from physical memory to virtual memory on disk. The higher the value, the more aggressively inactive processes are swapped out from physical memory. The lower the value, the less they are swapped, forcing filesystem buffers to be emptied. It is an important kernel parameter for Kafka because the more memory allocated to the swap space, the less memory can be allocated to the page cache. Cloudera recommends to set `vm.swappiness` value to `1`.

- To check memory swapped to disk, run `vmstat` and look for the swap columns.

Kafka heavily relies on disk I/O performance. `vm.dirty_ratio` and `vm.dirty_background_ratio` are kernel parameters that control how often dirty pages are flushed to disk. Higher `vm.dirty_ratio` results in less frequent flushes to disk.

- To display the actual number of dirty pages in the system, run `egrep "dirty|writeback" /proc/vmstat`

Networking Parameters

Kafka is designed to handle a huge amount of network traffic. By default, the Linux kernel is not tuned for this scenario. The following kernel settings may need to be tuned based on use case or specific Kafka workload:

- `net.core.wmem_default`: Default send socket buffer size.
- `net.core.rmem_default`: Default receive socket buffer size.
- `net.core.wmem_max`: Maximum send socket buffer size.
- `net.core.rmem_max`: Maximum receive socket buffer size.
- `net.ipv4.tcp_wmem`: Memory reserved for TCP send buffers.
- `net.ipv4.tcp_rmem`: Memory reserved for TCP receive buffers.
- `net.ipv4.tcp_window_scaling`: TCP Window Scaling option.
- `net.ipv4.tcp_max_syn_backlog`: Maximum number of outstanding TCP SYN requests (connection requests).
- `net.core.netdev_max_backlog`: Maximum number of queued packets on the kernel input side (useful to deal with spike of network requests).

To specify the parameters, you can use Cloudera Enterprise Reference Architecture as a guideline.

Configuring JMX Ephemeral Ports

Kafka uses two high-numbered ephemeral ports for JMX. These ports are listed when you view `netstat -anp` information for the Kafka broker process.

- You can change the number for the first port by adding a command similar to the following to the field Additional Broker Java Options (`broker_java_opts`) in Cloudera Manager.

```
-Dcom.sun.management.jmxremote.rmi.port=port
```

- The JMX_PORT configuration maps to `com.sun.management.jmxremote.port` by default.

  To access JMX via JConsole, run `jconsole ${BROKER_HOST}:9393`

- The second ephemeral port used for JMX communication is implemented for the JRMP protocol and cannot be changed.

Kafka-ZooKeeper Performance Tuning

Kafka uses Zookeeper to store metadata information about topics, partitions, brokers and system coordination (such as membership statuses). Unavailability or slowness of Zookeeper makes the Kafka cluster unstable, and Kafka brokers do not automatically recover from it. Cloudera recommends to use a 3-5 machines Zookeeper ensemble solely dedicated to Kafka (co-location of applications can cause unwanted service disturbances).

- `zookeeper.session.timeout.ms` is a setting for Kafka that specifies how long Zookeeper shall wait for heartbeat messages before it considers the client (the Kafka broker) unavailable. If this happens, metadata information about partition leadership owned by the broker will be reassigned. If this setting is too high, then it might take a long time for the system to detect a broker failure. On the other hand, if it is set to too small, it might result in frequent leadership reassignments.
- `jute.maxbuffer` is a crucial Java system property for both Kafka and Zookeeper. It controls the maximum size of the data a znode can contain. The default value, one megabyte, might be increased for certain production use cases.
- There are cases where Zookeeper can require more connections. In those cases, it is recommended to increase the `maxClientCnxns` parameter in Zookeeper.
- Note that old Kafka consumers store consumer offset commits in Zookeeper (deprecated). It is recommended to use new consumers that store offsets in internal Kafka topics (reduces load on Zookeeper).

Kafka Reference

Metrics Reference

In addition to these metrics, many aggregate metrics are available. If an entity type has parents defined, you can formulate all possible aggregate metrics using the formula `base_metric_across_parents`.

In addition, metrics for aggregate totals can be formed by adding the prefix total_ to the front of the metric name.

Use the type-ahead feature in the Cloudera Manager chart browser to find the exact aggregate metric name, in case the plural form does not end in "s". For example, the following metric names may be valid for Kafka:

- `alerts_rate_across_clusters`
- `total_alerts_rate_across_clusters`

Some metrics, such as `alerts_rate`, apply to nearly every metric context. Others only apply to a certain service or role.

For more information about metrics the Cloudera Manager, see Cloudera Manager Metrics and Metric Aggregation.

Note: The following sections are identical to the metrics listed with Cloudera Manager. Be sure to scroll horizontally to see the full content in each table.

Base Metrics

Metric Name	Description	Unit	Parents	CDH Version
`alerts_rate`	The number of alerts.	events per second	cluster	CDH 5, CDH 6
`events_critical_rate`	The number of critical events.	events per second	cluster	CDH 5, CDH 6
`events_important_rate`	The number of important events.	events per second	cluster	CDH 5, CDH 6
`events_informational_rate`	The number of informational events.	events per second	cluster	CDH 5, CDH 6
`health_bad_rate`	Percentage of Time with Bad Health	seconds per second	cluster	CDH 5, CDH 6
`health_concerning_rate`	Percentage of Time with Concerning Health	seconds per second	cluster	CDH 5, CDH 6
`health_disabled_rate`	Percentage of Time with Disabled Health	seconds per second	cluster	CDH 5, CDH 6
`health_good_rate`	Percentage of Time with Good Health	seconds per second	cluster	CDH 5, CDH 6
`health_unknown_rate`	Percentage of Time with Unknown Health	seconds per second	cluster	CDH 5, CDH 6

Broker Metrics

Metric Name	Description	Unit	Parents	CDH Version
alerts_rate	The number of alerts.	events per second	cluster, kafka, rack	CDH 5, CDH 6
cgroup_cpu_system_rate	CPU usage of the role's cgroup	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
cgroup_cpu_user_rate	User Space CPU usage of the role's cgroup	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
cgroup_mem_page_cache	Page cache usage of the role's cgroup	bytes	cluster, kafka, rack	CDH 5, CDH 6
cgroup_mem_rss	Resident memory of the role's cgroup	bytes	cluster, kafka, rack	CDH 5, CDH 6
cgroup_mem_swap	Swap usage of the role's cgroup	bytes	cluster, kafka, rack	CDH 5, CDH 6
cgroup_read_bytes_rate	Bytes read from all disks by the role's cgroup	bytes per second	cluster, kafka, rack	CDH 5, CDH 6
cgroup_read_ios_rate	Number of read I/O operations from all disks by the role's cgroup	ios per second	cluster, kafka, rack	CDH 5, CDH 6
cgroup_write_bytes_rate	Bytes written to all disks by the role's cgroup	bytes per second	cluster, kafka, rack	CDH 5, CDH 6
cgroup_write_ios_rate	Number of write I/O operations to all disks by the role's cgroup	ios per second	cluster, kafka, rack	CDH 5, CDH 6
cpu_system_rate	Total System CPU	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
cpu_user_rate	Total CPU user time	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
events_critical_rate	The number of critical events.	events per second	cluster, kafka, rack	CDH 5, CDH 6
events_important_rate	The number of important events.	events per second	cluster, kafka, rack	CDH 5, CDH 6
events_informational_rate	The number of informational events.	events per second	cluster, kafka, rack	CDH 5, CDH 6
fd_max	Maximum number of file descriptors	file descriptors	cluster, kafka, rack	CDH 5, CDH 6
fd_open	Open file descriptors.	file descriptors	cluster, kafka, rack	CDH 5, CDH 6
health_bad_rate	Percentage of Time with Bad Health	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
health_concerning_rate	Percentage of Time with Concerning Health	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
health_disabled_rate	Percentage of Time with Disabled Health	seconds per second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
health_good_rate	Percentage of Time with Good Health	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
health_unknown_rate	Percentage of Time with Unknown Health	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_active_controller	Will be 1 if this broker is the active controller, 0 otherwise	message.units.controller	cluster, kafka, rack	CDH 5, CDH 6
kafka_broker_state	The state the broker is in. • 0 = NotRunning • 1 = Starting • 2 = RecoveringFromUncleanShutdown • 3 = RunningAsBroker • 4 = RunningAsController • 6 = PendingControlledShutdown • 7 = BrokerShuttingDown	message.units.state	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_fetched_15min_rate	Amount of data consumers fetched from this topic on this broker: 15 Min Rate	bytes per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_fetched_1min_rate	Amount of data consumers fetched from this topic on this broker: 1 Min Rate	bytes per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_fetched_5min_rate	Amount of data consumers fetched from this topic on this broker: 5 Min Rate	bytes per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_fetched_avg_rate	Amount of data consumers fetched from this topic on this broker: Avg Rate	bytes per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_fetched_rate	Amount of data consumers fetched from this topic on this broker	bytes per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_received_15min_rate	Amount of data written to topic on this broker: 15 Min Rate	bytes per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_received_1min_rate	Amount of data written to topic on this broker: 1 Min Rate	bytes per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_received_5min_rate	Amount of data written to topic on this broker: 5 Min Rate	bytes per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_received_avg_rate	Amount of data written to topic on this broker: Avg Rate	bytes per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_bytes_received_rate	Amount of data written to topic on this broker	bytes per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_rejected_15min_rate	Amount of data in messages rejected by broker for this topic: 15 Min Rate	bytes per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_rejected_1min_rate	Amount of data in messages rejected by broker for this topic: 1 Min Rate	bytes per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_rejected_5min_rate	Amount of data in messages rejected by broker for this topic: 5 Min Rate	bytes per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_rejected_avg_rate	Amount of data in messages rejected by broker for this topic: Avg Rate	bytes per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_bytes_rejected_rate	Amount of data in messages rejected by broker for this topic	bytes per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_expires_15min_rate	Number of expired delayed consumer fetch requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_expires_1min_rate	Number of expired delayed consumer fetch requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_expires_5min_rate	Number of expired delayed consumer fetch requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_expires_avg_rate	Number of expired delayed consumer fetch requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_expires_rate	Number of expired delayed consumer fetch requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_local_time_75th_percentile	Local Time spent in responding to ConsumerMetadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_local_time_999th_percentile	Local Time spent in responding to ConsumerMetadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_local_time_99th_percentile	Local Time spent in responding to ConsumerMetadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_local_time_avg	Local Time spent in responding to ConsumerMetadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_local_time_max	Local Time spent in responding to ConsumerMetadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_consumer_metadata_local_time_median	Local Time spent in responding to ConsumerMetadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_local_time_min	Local Time spent in responding to ConsumerMetadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_local_time_rate	Local Time spent in responding to ConsumerMetadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_local_time_stddev	Local Time spent in responding to ConsumerMetadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_remote_time_75th_percentile	Remote Time spent in responding to ConsumerMetadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_remote_time_999th_percentile	Remote Time spent in responding to ConsumerMetadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_remote_time_99th_percentile	Remote Time spent in responding to ConsumerMetadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_remote_time_avg	Remote Time spent in responding to ConsumerMetadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_remote_time_max	Remote Time spent in responding to ConsumerMetadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_remote_time_median	Remote Time spent in responding to ConsumerMetadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_remote_time_min	Remote Time spent in responding to ConsumerMetadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_remote_time_rate	Remote Time spent in responding to ConsumerMetadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_remote_time_stddev	Remote Time spent in responding to ConsumerMetadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_consumer_metadata_request_queue_time_75th_percentile	Request Queue Time spent in responding to ConsumerMetadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_request_queue_time_999th_percentile	Request Queue Time spent in responding to ConsumerMetadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_request_queue_time_99th_percentile	Request Queue Time spent in responding to ConsumerMetadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_request_queue_time_avg	Request Queue Time spent in responding to ConsumerMetadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_request_queue_time_max	Request Queue Time spent in responding to ConsumerMetadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_request_queue_time_median	Request Queue Time spent in responding to ConsumerMetadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_request_queue_time_min	Request Queue Time spent in responding to ConsumerMetadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_request_queue_time_rate	Request Queue Time spent in responding to ConsumerMetadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_request_queue_time_stddev	Request Queue Time spent in responding to ConsumerMetadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_requests_15min_rate	Number of ConsumerMetadata requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_requests_1min_rate	Number of ConsumerMetadata requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_requests_5min_rate	Number of ConsumerMetadata requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_consumer_metadata_requests_avg_rate	Number of ConsumerMetadata requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_requests_rate	Number of ConsumerMetadata requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_queue_time_75th_percentile	Response Queue Time spent in responding to ConsumerMetadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_queue_time_999th_percentile	Response Queue Time spent in responding to ConsumerMetadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_queue_time_99th_percentile	Response Queue Time spent in responding to ConsumerMetadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_queue_time_avg	Response Queue Time spent in responding to ConsumerMetadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_queue_time_max	Response Queue Time spent in responding to ConsumerMetadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_queue_time_median	Response Queue Time spent in responding to ConsumerMetadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_queue_time_min	Response Queue Time spent in responding to ConsumerMetadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_queue_time_rate	Response Queue Time spent in responding to ConsumerMetadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_queue_time_stddev	Response Queue Time spent in responding to ConsumerMetadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_send_time_75th_percentile	Response Send Time spent in responding to ConsumerMetadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_consumer_metadata_response_send_time_999th_percentile	Response Send Time spent in responding to ConsumerMetadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_send_time_99th_percentile	Response Send Time spent in responding to ConsumerMetadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_send_time_avg	Response Send Time spent in responding to ConsumerMetadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_send_time_max	Response Send Time spent in responding to ConsumerMetadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_send_time_median	Response Send Time spent in responding to ConsumerMetadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_send_time_min	Response Send Time spent in responding to ConsumerMetadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_send_time_rate	Response Send Time spent in responding to ConsumerMetadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_response_send_time_stddev	Response Send Time spent in responding to ConsumerMetadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_total_time_75th_percentile	Total Time spent in responding to ConsumerMetadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_total_time_999th_percentile	Total Time spent in responding to ConsumerMetadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_total_time_99th_percentile	Total Time spent in responding to ConsumerMetadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_total_time_avg	Total Time spent in responding to ConsumerMetadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_total_time_max	Total Time spent in responding to ConsumerMetadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_consumer_metadata_total_time_median	Total Time spent in responding to ConsumerMetadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_total_time_min	Total Time spent in responding to ConsumerMetadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_total_time_rate	Total Time spent in responding to ConsumerMetadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_consumer_metadata_total_time_stddev	Total Time spent in responding to ConsumerMetadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_local_time_75th_percentile	Local Time spent in responding to ControlledShutdown requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_local_time_999th_percentile	Local Time spent in responding to ControlledShutdown requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_local_time_99th_percentile	Local Time spent in responding to ControlledShutdown requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_local_time_avg	Local Time spent in responding to ControlledShutdown requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_local_time_max	Local Time spent in responding to ControlledShutdown requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_local_time_median	Local Time spent in responding to ControlledShutdown requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_local_time_min	Local Time spent in responding to ControlledShutdown requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_local_time_rate	Local Time spent in responding to ControlledShutdown requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_local_time_stddev	Local Time spent in responding to ControlledShutdown requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_remote_time_75th_percentile	Remote Time spent in responding to ControlledShutdown requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_controlled_shutdown_remote_time_999th_percentile	Remote Time spent in responding to ControlledShutdown requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_remote_time_99th_percentile	Remote Time spent in responding to ControlledShutdown requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_remote_time_avg	Remote Time spent in responding to ControlledShutdown requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_remote_time_max	Remote Time spent in responding to ControlledShutdown requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_remote_time_median	Remote Time spent in responding to ControlledShutdown requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_remote_time_min	Remote Time spent in responding to ControlledShutdown requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_remote_time_rate	Remote Time spent in responding to ControlledShutdown requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_remote_time_stddev	Remote Time spent in responding to ControlledShutdown requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_request_queue_time_75th_percentile	Request Queue Time spent in responding to ControlledShutdown requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_request_queue_time_999th_percentile	Request Queue Time spent in responding to ControlledShutdown requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_request_queue_time_99th_percentile	Request Queue Time spent in responding to ControlledShutdown requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_request_queue_time_avg	Request Queue Time spent in responding to ControlledShutdown requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_controlled_shutdown_request_queue_time_max	Request Queue Time spent in responding to ControlledShutdown requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_request_queue_time_median	Request Queue Time spent in responding to ControlledShutdown requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_request_queue_time_min	Request Queue Time spent in responding to ControlledShutdown requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_request_queue_time_rate	Request Queue Time spent in responding to ControlledShutdown requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_request_queue_time_stddev	Request Queue Time spent in responding to ControlledShutdown requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_requests_15min_rate	Number of ControlledShutdown requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_requests_1min_rate	Number of ControlledShutdown requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_requests_5min_rate	Number of ControlledShutdown requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_requests_avg_rate	Number of ControlledShutdown requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_requests_rate	Number of ControlledShutdown requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_queue_time_75th_percentile	Response Queue Time spent in responding to ControlledShutdown requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_queue_time_999th_percentile	Response Queue Time spent in responding to ControlledShutdown requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_queue_time_99th_percentile	Response Queue Time spent in responding to ControlledShutdown requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_controlled_shutdown_response_queue_time_avg	Response Queue Time spent in responding to ControlledShutdown requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_queue_time_max	Response Queue Time spent in responding to ControlledShutdown requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_queue_time_median	Response Queue Time spent in responding to ControlledShutdown requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_queue_time_min	Response Queue Time spent in responding to ControlledShutdown requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_queue_time_rate	Response Queue Time spent in responding to ControlledShutdown requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_queue_time_stddev	Response Queue Time spent in responding to ControlledShutdown requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_send_time_75th_percentile	Response Send Time spent in responding to ControlledShutdown requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_send_time_999th_percentile	Response Send Time spent in responding to ControlledShutdown requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_send_time_99th_percentile	Response Send Time spent in responding to ControlledShutdown requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_send_time_avg	Response Send Time spent in responding to ControlledShutdown requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_send_time_max	Response Send Time spent in responding to ControlledShutdown requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_send_time_median	Response Send Time spent in responding to ControlledShutdown requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_controlled_shutdown_response_send_time_min	Response Send Time spent in responding to ControlledShutdown requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_send_time_rate	Response Send Time spent in responding to ControlledShutdown requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_response_send_time_stddev	Response Send Time spent in responding to ControlledShutdown requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_total_time_75th_percentile	Total Time spent in responding to ControlledShutdown requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_total_time_999th_percentile	Total Time spent in responding to ControlledShutdown requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_total_time_99th_percentile	Total Time spent in responding to ControlledShutdown requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_total_time_avg	Total Time spent in responding to ControlledShutdown requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_total_time_max	Total Time spent in responding to ControlledShutdown requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_total_time_median	Total Time spent in responding to ControlledShutdown requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_total_time_min	Total Time spent in responding to ControlledShutdown requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_total_time_rate	Total Time spent in responding to ControlledShutdown requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_controlled_shutdown_total_time_stddev	Total Time spent in responding to ControlledShutdown requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_daemon_thread_count	JVM daemon thread count	threads	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_local_time_75th_percentile	Local Time spent in responding to FetchConsumer requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_fetch_consumer_local_time_999th_percentile	Local Time spent in responding to FetchConsumer requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_local_time_99th_percentile	Local Time spent in responding to FetchConsumer requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_local_time_avg	Local Time spent in responding to FetchConsumer requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_local_time_max	Local Time spent in responding to FetchConsumer requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_local_time_median	Local Time spent in responding to FetchConsumer requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_local_time_min	Local Time spent in responding to FetchConsumer requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_local_time_rate	Local Time spent in responding to FetchConsumer requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_local_time_stddev	Local Time spent in responding to FetchConsumer requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_remote_time_75th_percentile	Remote Time spent in responding to FetchConsumer requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_remote_time_999th_percentile	Remote Time spent in responding to FetchConsumer requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_remote_time_99th_percentile	Remote Time spent in responding to FetchConsumer requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_remote_time_avg	Remote Time spent in responding to FetchConsumer requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_remote_time_max	Remote Time spent in responding to FetchConsumer requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_remote_time_median	Remote Time spent in responding to FetchConsumer requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_remote_time_min	Remote Time spent in responding to FetchConsumer requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_remote_time_rate	Remote Time spent in responding to FetchConsumer requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_fetch_consumer_remote_time_stddev	Remote Time spent in responding to FetchConsumer requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_request_queue_time_75th_percentile	Request Queue Time spent in responding to FetchConsumer requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_request_queue_time_999th_percentile	Request Queue Time spent in responding to FetchConsumer requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_request_queue_time_99th_percentile	Request Queue Time spent in responding to FetchConsumer requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_request_queue_time_avg	Request Queue Time spent in responding to FetchConsumer requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_request_queue_time_max	Request Queue Time spent in responding to FetchConsumer requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_request_queue_time_median	Request Queue Time spent in responding to FetchConsumer requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_request_queue_time_min	Request Queue Time spent in responding to FetchConsumer requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_request_queue_time_rate	Request Queue Time spent in responding to FetchConsumer requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_request_queue_time_stddev	Request Queue Time spent in responding to FetchConsumer requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_requests_15min_rate	Number of FetchConsumer requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_requests_1min_rate	Number of FetchConsumer requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_requests_5min_rate	Number of FetchConsumer requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_requests_avg_rate	Number of FetchConsumer requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_requests_rate	Number of FetchConsumer requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_fetch_consumer_response_queue_time_75th_percentile	Response Queue Time spent in responding to FetchConsumer requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_queue_time_999th_percentile	Response Queue Time spent in responding to FetchConsumer requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_queue_time_99th_percentile	Response Queue Time spent in responding to FetchConsumer requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_queue_time_avg	Response Queue Time spent in responding to FetchConsumer requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_queue_time_max	Response Queue Time spent in responding to FetchConsumer requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_queue_time_median	Response Queue Time spent in responding to FetchConsumer requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_queue_time_min	Response Queue Time spent in responding to FetchConsumer requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_queue_time_rate	Response Queue Time spent in responding to FetchConsumer requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_queue_time_stddev	Response Queue Time spent in responding to FetchConsumer requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_send_time_75th_percentile	Response Send Time spent in responding to FetchConsumer requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_send_time_999th_percentile	Response Send Time spent in responding to FetchConsumer requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_send_time_99th_percentile	Response Send Time spent in responding to FetchConsumer requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_send_time_avg	Response Send Time spent in responding to FetchConsumer requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_send_time_max	Response Send Time spent in responding to FetchConsumer requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_send_time_median	Response Send Time spent in responding to FetchConsumer requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_fetch_consumer_response_send_time_min	Response Send Time spent in responding to FetchConsumer requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_send_time_rate	Response Send Time spent in responding to FetchConsumer requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_response_send_time_stddev	Response Send Time spent in responding to FetchConsumer requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_total_time_75th_percentile	Total Time spent in responding to FetchConsumer requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_total_time_999th_percentile	Total Time spent in responding to FetchConsumer requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_total_time_99th_percentile	Total Time spent in responding to FetchConsumer requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_total_time_avg	Total Time spent in responding to FetchConsumer requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_total_time_max	Total Time spent in responding to FetchConsumer requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_total_time_median	Total Time spent in responding to FetchConsumer requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_total_time_min	Total Time spent in responding to FetchConsumer requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_total_time_rate	Total Time spent in responding to FetchConsumer requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_consumer_total_time_stddev	Total Time spent in responding to FetchConsumer requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_local_time_75th_percentile	Local Time spent in responding to FetchFollower requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_local_time_999th_percentile	Local Time spent in responding to FetchFollower requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_local_time_99th_percentile	Local Time spent in responding to FetchFollower requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_local_time_avg	Local Time spent in responding to FetchFollower requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_fetch_follower_local_time_max	Local Time spent in responding to FetchFollower requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_local_time_median	Local Time spent in responding to FetchFollower requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_local_time_min	Local Time spent in responding to FetchFollower requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_local_time_rate	Local Time spent in responding to FetchFollower requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_local_time_stddev	Local Time spent in responding to FetchFollower requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_remote_time_75th_percentile	Remote Time spent in responding to FetchFollower requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_remote_time_999th_percentile	Remote Time spent in responding to FetchFollower requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_remote_time_99th_percentile	Remote Time spent in responding to FetchFollower requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_remote_time_avg	Remote Time spent in responding to FetchFollower requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_remote_time_max	Remote Time spent in responding to FetchFollower requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_remote_time_median	Remote Time spent in responding to FetchFollower requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_remote_time_min	Remote Time spent in responding to FetchFollower requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_remote_time_rate	Remote Time spent in responding to FetchFollower requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_remote_time_stddev	Remote Time spent in responding to FetchFollower requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_request_queue_time_75th_percentile	Request Queue Time spent in responding to FetchFollower requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_request_queue_time_999th_percentile	Request Queue Time spent in responding to FetchFollower requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_fetch_follower_request_queue_time_99th_percentile	Request Queue Time spent in responding to FetchFollower requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_request_queue_time_avg	Request Queue Time spent in responding to FetchFollower requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_request_queue_time_max	Request Queue Time spent in responding to FetchFollower requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_request_queue_time_median	Request Queue Time spent in responding to FetchFollower requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_request_queue_time_min	Request Queue Time spent in responding to FetchFollower requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_request_queue_time_rate	Request Queue Time spent in responding to FetchFollower requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_request_queue_time_stddev	Request Queue Time spent in responding to FetchFollower requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_requests_15min_rate	Number of FetchFollower requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_requests_1min_rate	Number of FetchFollower requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_requests_5min_rate	Number of FetchFollower requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_requests_avg_rate	Number of FetchFollower requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_requests_rate	Number of FetchFollower requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_queue_time_75th_percentile	Response Queue Time spent in responding to FetchFollower requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_queue_time_999th_percentile	Response Queue Time spent in responding to FetchFollower requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_queue_time_99th_percentile	Response Queue Time spent in responding to FetchFollower requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_fetch_follower_response_queue_time_avg	Response Queue Time spent in responding to FetchFollower requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_queue_time_max	Response Queue Time spent in responding to FetchFollower requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_queue_time_median	Response Queue Time spent in responding to FetchFollower requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_queue_time_min	Response Queue Time spent in responding to FetchFollower requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_queue_time_rate	Response Queue Time spent in responding to FetchFollower requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_queue_time_stddev	Response Queue Time spent in responding to FetchFollower requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_send_time_75th_percentile	Response Send Time spent in responding to FetchFollower requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_send_time_999th_percentile	Response Send Time spent in responding to FetchFollower requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_send_time_99th_percentile	Response Send Time spent in responding to FetchFollower requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_send_time_avg	Response Send Time spent in responding to FetchFollower requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_send_time_max	Response Send Time spent in responding to FetchFollower requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_send_time_median	Response Send Time spent in responding to FetchFollower requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_send_time_min	Response Send Time spent in responding to FetchFollower requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_send_time_rate	Response Send Time spent in responding to FetchFollower requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_response_send_time_stddev	Response Send Time spent in responding to FetchFollower requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_fetch_follower_total_time_75th_percentile	Total Time spent in responding to FetchFollower requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_total_time_999th_percentile	Total Time spent in responding to FetchFollower requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_total_time_99th_percentile	Total Time spent in responding to FetchFollower requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_total_time_avg	Total Time spent in responding to FetchFollower requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_total_time_max	Total Time spent in responding to FetchFollower requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_total_time_median	Total Time spent in responding to FetchFollower requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_total_time_min	Total Time spent in responding to FetchFollower requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_total_time_rate	Total Time spent in responding to FetchFollower requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_follower_total_time_stddev	Total Time spent in responding to FetchFollower requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_local_time_75th_percentile	Local Time spent in responding to Fetch requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_local_time_999th_percentile	Local Time spent in responding to Fetch requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_local_time_99th_percentile	Local Time spent in responding to Fetch requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_local_time_avg	Local Time spent in responding to Fetch requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_local_time_max	Local Time spent in responding to Fetch requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_local_time_median	Local Time spent in responding to Fetch requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_local_time_min	Local Time spent in responding to Fetch requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_local_time_rate	Local Time spent in responding to Fetch requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_fetch_local_time_stddev	Local Time spent in responding to Fetch requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_purgatory_delayed_requests	Number of requests delayed in the fetch purgatory	requests	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_purgatory_size	Requests waiting in the fetch purgatory. This depends on value of fetch.wait.max.ms in the consumer	requests	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_remote_time_75th_percentile	Remote Time spent in responding to Fetch requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_remote_time_999th_percentile	Remote Time spent in responding to Fetch requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_remote_time_99th_percentile	Remote Time spent in responding to Fetch requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_remote_time_avg	Remote Time spent in responding to Fetch requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_remote_time_max	Remote Time spent in responding to Fetch requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_remote_time_median	Remote Time spent in responding to Fetch requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_remote_time_min	Remote Time spent in responding to Fetch requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_remote_time_rate	Remote Time spent in responding to Fetch requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_remote_time_stddev	Remote Time spent in responding to Fetch requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_failures_15min_rate	Number of data read requests from consumers that brokers failed to process for this topic: 15 Min Rate	message.units.fetch_requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_failures_1min_rate	Number of data read requests from consumers that brokers failed to process for this topic: 1 Min Rate	message.units.fetch_requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_fetch_request_failures_5min_rate	Number of data read requests from consumers that brokers failed to process for this topic: 5 Min Rate	message.units.fetch_requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_failures_avg_rate	Number of data read requests from consumers that brokers failed to process for this topic: Avg Rate	message.units.fetch_requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_failures_rate	Number of data read requests from consumers that brokers failed to process for this topic	message.units.fetch_requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_queue_time_75th_percentile	Request Queue Time spent in responding to Fetch requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_queue_time_999th_percentile	Request Queue Time spent in responding to Fetch requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_queue_time_99th_percentile	Request Queue Time spent in responding to Fetch requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_queue_time_avg	Request Queue Time spent in responding to Fetch requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_queue_time_max	Request Queue Time spent in responding to Fetch requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_queue_time_median	Request Queue Time spent in responding to Fetch requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_queue_time_min	Request Queue Time spent in responding to Fetch requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_queue_time_rate	Request Queue Time spent in responding to Fetch requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_request_queue_time_stddev	Request Queue Time spent in responding to Fetch requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_requests_15min_rate	Number of Fetch requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_requests_1min_rate	Number of Fetch requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_fetch_requests_5min_rate	Number of Fetch requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_requests_avg_rate	Number of Fetch requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_requests_rate	Number of Fetch requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_queue_time_75th_percentile	Response Queue Time spent in responding to Fetch requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_queue_time_999th_percentile	Response Queue Time spent in responding to Fetch requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_queue_time_99th_percentile	Response Queue Time spent in responding to Fetch requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_queue_time_avg	Response Queue Time spent in responding to Fetch requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_queue_time_max	Response Queue Time spent in responding to Fetch requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_queue_time_median	Response Queue Time spent in responding to Fetch requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_queue_time_min	Response Queue Time spent in responding to Fetch requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_queue_time_rate	Response Queue Time spent in responding to Fetch requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_queue_time_stddev	Response Queue Time spent in responding to Fetch requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_send_time_75th_percentile	Response Send Time spent in responding to Fetch requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_send_time_999th_percentile	Response Send Time spent in responding to Fetch requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_send_time_99th_percentile	Response Send Time spent in responding to Fetch requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_fetch_response_send_time_avg	Response Send Time spent in responding to Fetch requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_send_time_max	Response Send Time spent in responding to Fetch requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_send_time_median	Response Send Time spent in responding to Fetch requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_send_time_min	Response Send Time spent in responding to Fetch requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_send_time_rate	Response Send Time spent in responding to Fetch requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_response_send_time_stddev	Response Send Time spent in responding to Fetch requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_total_time_75th_percentile	Total Time spent in responding to Fetch requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_total_time_999th_percentile	Total Time spent in responding to Fetch requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_total_time_99th_percentile	Total Time spent in responding to Fetch requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_total_time_avg	Total Time spent in responding to Fetch requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_total_time_max	Total Time spent in responding to Fetch requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_total_time_median	Total Time spent in responding to Fetch requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_total_time_min	Total Time spent in responding to Fetch requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_total_time_rate	Total Time spent in responding to Fetch requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_fetch_total_time_stddev	Total Time spent in responding to Fetch requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_follower_expires_15min_rate	Number of expired delayed follower fetch requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_follower_expires_1min_rate	Number of expired delayed follower fetch requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_follower_expires_5min_rate	Number of expired delayed follower fetch requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_follower_expires_avg_rate	Number of expired delayed follower fetch requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_follower_expires_rate	Number of expired delayed follower fetch requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_local_time_75th_percentile	Local Time spent in responding to Heartbeat requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_local_time_999th_percentile	Local Time spent in responding to Heartbeat requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_local_time_99th_percentile	Local Time spent in responding to Heartbeat requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_local_time_avg	Local Time spent in responding to Heartbeat requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_local_time_max	Local Time spent in responding to Heartbeat requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_local_time_median	Local Time spent in responding to Heartbeat requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_local_time_min	Local Time spent in responding to Heartbeat requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_local_time_rate	Local Time spent in responding to Heartbeat requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_local_time_stddev	Local Time spent in responding to Heartbeat requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_remote_time_75th_percentile	Remote Time spent in responding to Heartbeat requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_remote_time_999th_percentile	Remote Time spent in responding to Heartbeat requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_remote_time_99th_percentile	Remote Time spent in responding to Heartbeat requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_heartbeat_remote_time_avg	Remote Time spent in responding to Heartbeat requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_remote_time_max	Remote Time spent in responding to Heartbeat requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_remote_time_median	Remote Time spent in responding to Heartbeat requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_remote_time_min	Remote Time spent in responding to Heartbeat requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_remote_time_rate	Remote Time spent in responding to Heartbeat requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_remote_time_stddev	Remote Time spent in responding to Heartbeat requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_request_queue_time_75th_percentile	Request Queue Time spent in responding to Heartbeat requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_request_queue_time_999th_percentile	Request Queue Time spent in responding to Heartbeat requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_request_queue_time_99th_percentile	Request Queue Time spent in responding to Heartbeat requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_request_queue_time_avg	Request Queue Time spent in responding to Heartbeat requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_request_queue_time_max	Request Queue Time spent in responding to Heartbeat requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_request_queue_time_median	Request Queue Time spent in responding to Heartbeat requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_request_queue_time_min	Request Queue Time spent in responding to Heartbeat requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_request_queue_time_rate	Request Queue Time spent in responding to Heartbeat requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_request_queue_time_stddev	Request Queue Time spent in responding to Heartbeat requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_heartbeat_requests_15min_rate	Number of Heartbeat requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_requests_1min_rate	Number of Heartbeat requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_requests_5min_rate	Number of Heartbeat requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_requests_avg_rate	Number of Heartbeat requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_requests_rate	Number of Heartbeat requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_queue_time_75th_percentile	Response Queue Time spent in responding to Heartbeat requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_queue_time_999th_percentile	Response Queue Time spent in responding to Heartbeat requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_queue_time_99th_percentile	Response Queue Time spent in responding to Heartbeat requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_queue_time_avg	Response Queue Time spent in responding to Heartbeat requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_queue_time_max	Response Queue Time spent in responding to Heartbeat requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_queue_time_median	Response Queue Time spent in responding to Heartbeat requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_queue_time_min	Response Queue Time spent in responding to Heartbeat requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_queue_time_rate	Response Queue Time spent in responding to Heartbeat requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_queue_time_stddev	Response Queue Time spent in responding to Heartbeat requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_send_time_75th_percentile	Response Send Time spent in responding to Heartbeat requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_heartbeat_response_send_time_999th_percentile	Response Send Time spent in responding to Heartbeat requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_send_time_99th_percentile	Response Send Time spent in responding to Heartbeat requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_send_time_avg	Response Send Time spent in responding to Heartbeat requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_send_time_max	Response Send Time spent in responding to Heartbeat requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_send_time_median	Response Send Time spent in responding to Heartbeat requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_send_time_min	Response Send Time spent in responding to Heartbeat requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_send_time_rate	Response Send Time spent in responding to Heartbeat requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_response_send_time_stddev	Response Send Time spent in responding to Heartbeat requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_total_time_75th_percentile	Total Time spent in responding to Heartbeat requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_total_time_999th_percentile	Total Time spent in responding to Heartbeat requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_total_time_99th_percentile	Total Time spent in responding to Heartbeat requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_total_time_avg	Total Time spent in responding to Heartbeat requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_total_time_max	Total Time spent in responding to Heartbeat requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_total_time_median	Total Time spent in responding to Heartbeat requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_total_time_min	Total Time spent in responding to Heartbeat requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_heartbeat_total_time_rate	Total Time spent in responding to Heartbeat requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_heartbeat_total_time_stddev	Total Time spent in responding to Heartbeat requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_isr_expands_15min_rate	Number of times ISR for a partition expanded: 15 Min Rate	message.units. expansions per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_isr_expands_1min_rate	Number of times ISR for a partition expanded: 1 Min Rate	message.units. expansions per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_isr_expands_5min_rate	Number of times ISR for a partition expanded: 5 Min Rate	message.units. expansions per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_isr_expands_avg_rate	Number of times ISR for a partition expanded: Avg Rate	message.units. expansions per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_isr_expands_rate	Number of times ISR for a partition expanded	message.units. expansions per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_isr_shrinks_15min_rate	Number of times ISR for a partition shrank: 15 Min Rate	message.units. shrinks per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_isr_shrinks_1min_rate	Number of times ISR for a partition shrank: 1 Min Rate	message.units. shrinks per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_isr_shrinks_5min_rate	Number of times ISR for a partition shrank: 5 Min Rate	message.units. shrinks per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_isr_shrinks_avg_rate	Number of times ISR for a partition shrank: Avg Rate	message.units. shrinks per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_isr_shrinks_rate	Number of times ISR for a partition shrank	message.units. shrinks per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_local_time_75th_percentile	Local Time spent in responding to JoinGroup requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_local_time_999th_percentile	Local Time spent in responding to JoinGroup requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_join_group_local_time_99th_percentile	Local Time spent in responding to JoinGroup requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_local_time_avg	Local Time spent in responding to JoinGroup requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_local_time_max	Local Time spent in responding to JoinGroup requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_local_time_median	Local Time spent in responding to JoinGroup requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_local_time_min	Local Time spent in responding to JoinGroup requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_local_time_rate	Local Time spent in responding to JoinGroup requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_local_time_stddev	Local Time spent in responding to JoinGroup requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_remote_time_75th_percentile	Remote Time spent in responding to JoinGroup requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_remote_time_999th_percentile	Remote Time spent in responding to JoinGroup requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_remote_time_99th_percentile	Remote Time spent in responding to JoinGroup requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_remote_time_avg	Remote Time spent in responding to JoinGroup requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_remote_time_max	Remote Time spent in responding to JoinGroup requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_remote_time_median	Remote Time spent in responding to JoinGroup requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_remote_time_min	Remote Time spent in responding to JoinGroup requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_remote_time_rate	Remote Time spent in responding to JoinGroup requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_remote_time_stddev	Remote Time spent in responding to JoinGroup requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_join_group_request_queue_time_75th_percentile	Request Queue Time spent in responding to JoinGroup requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_request_queue_time_999th_percentile	Request Queue Time spent in responding to JoinGroup requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_request_queue_time_99th_percentile	Request Queue Time spent in responding to JoinGroup requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_request_queue_time_avg	Request Queue Time spent in responding to JoinGroup requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_request_queue_time_max	Request Queue Time spent in responding to JoinGroup requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_request_queue_time_median	Request Queue Time spent in responding to JoinGroup requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_request_queue_time_min	Request Queue Time spent in responding to JoinGroup requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_request_queue_time_rate	Request Queue Time spent in responding to JoinGroup requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_request_queue_time_stddev	Request Queue Time spent in responding to JoinGroup requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_requests_15min_rate	Number of JoinGroup requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_requests_1min_rate	Number of JoinGroup requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_requests_5min_rate	Number of JoinGroup requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_requests_avg_rate	Number of JoinGroup requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_requests_rate	Number of JoinGroup requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_queue_time_75th_percentile	Response Queue Time spent in responding to JoinGroup requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_join_group_response_queue_time_999th_percentile	Response Queue Time spent in responding to JoinGroup requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_queue_time_99th_percentile	Response Queue Time spent in responding to JoinGroup requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_queue_time_avg	Response Queue Time spent in responding to JoinGroup requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_queue_time_max	Response Queue Time spent in responding to JoinGroup requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_queue_time_median	Response Queue Time spent in responding to JoinGroup requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_queue_time_min	Response Queue Time spent in responding to JoinGroup requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_queue_time_rate	Response Queue Time spent in responding to JoinGroup requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_queue_time_stddev	Response Queue Time spent in responding to JoinGroup requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_send_time_75th_percentile	Response Send Time spent in responding to JoinGroup requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_send_time_999th_percentile	Response Send Time spent in responding to JoinGroup requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_send_time_99th_percentile	Response Send Time spent in responding to JoinGroup requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_send_time_avg	Response Send Time spent in responding to JoinGroup requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_send_time_max	Response Send Time spent in responding to JoinGroup requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_send_time_median	Response Send Time spent in responding to JoinGroup requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_send_time_min	Response Send Time spent in responding to JoinGroup requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_join_group_response_send_time_rate	Response Send Time spent in responding to JoinGroup requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_response_send_time_stddev	Response Send Time spent in responding to JoinGroup requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_total_time_75th_percentile	Total Time spent in responding to JoinGroup requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_total_time_999th_percentile	Total Time spent in responding to JoinGroup requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_total_time_99th_percentile	Total Time spent in responding to JoinGroup requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_total_time_avg	Total Time spent in responding to JoinGroup requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_total_time_max	Total Time spent in responding to JoinGroup requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_total_time_median	Total Time spent in responding to JoinGroup requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_total_time_min	Total Time spent in responding to JoinGroup requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_total_time_rate	Total Time spent in responding to JoinGroup requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_join_group_total_time_stddev	Total Time spent in responding to JoinGroup requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_local_time_75th_percentile	Local Time spent in responding to LeaderAndIsr requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_local_time_999th_percentile	Local Time spent in responding to LeaderAndIsr requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_local_time_99th_percentile	Local Time spent in responding to LeaderAndIsr requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_local_time_avg	Local Time spent in responding to LeaderAndIsr requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_local_time_max	Local Time spent in responding to LeaderAndIsr requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_local_time_median	Local Time spent in responding to LeaderAndIsr requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_leader_and_isr_local_time_min	Local Time spent in responding to LeaderAndIsr requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_local_time_rate	Local Time spent in responding to LeaderAndIsr requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_local_time_stddev	Local Time spent in responding to LeaderAndIsr requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_remote_time_75th_percentile	Remote Time spent in responding to LeaderAndIsr requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_remote_time_999th_percentile	Remote Time spent in responding to LeaderAndIsr requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_remote_time_99th_percentile	Remote Time spent in responding to LeaderAndIsr requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_remote_time_avg	Remote Time spent in responding to LeaderAndIsr requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_remote_time_max	Remote Time spent in responding to LeaderAndIsr requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_remote_time_median	Remote Time spent in responding to LeaderAndIsr requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_remote_time_min	Remote Time spent in responding to LeaderAndIsr requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_remote_time_rate	Remote Time spent in responding to LeaderAndIsr requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_remote_time_stddev	Remote Time spent in responding to LeaderAndIsr requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_request_queue_time_75th_percentile	Request Queue Time spent in responding to LeaderAndIsr requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_request_queue_time_999th_percentile	Request Queue Time spent in responding to LeaderAndIsr requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_request_queue_time_99th_percentile	Request Queue Time spent in responding to LeaderAndIsr requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_leader_and_isr_request_queue_time_avg	Request Queue Time spent in responding to LeaderAndIsr requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_request_queue_time_max	Request Queue Time spent in responding to LeaderAndIsr requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_request_queue_time_median	Request Queue Time spent in responding to LeaderAndIsr requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_request_queue_time_min	Request Queue Time spent in responding to LeaderAndIsr requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_request_queue_time_rate	Request Queue Time spent in responding to LeaderAndIsr requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_request_queue_time_stddev	Request Queue Time spent in responding to LeaderAndIsr requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_requests_15min_rate	Number of LeaderAndIsr requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_requests_1min_rate	Number of LeaderAndIsr requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_requests_5min_rate	Number of LeaderAndIsr requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_requests_avg_rate	Number of LeaderAndIsr requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_requests_rate	Number of LeaderAndIsr requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_queue_time_75th_percentile	Response Queue Time spent in responding to LeaderAndIsr requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_queue_time_999th_percentile	Response Queue Time spent in responding to LeaderAndIsr requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_queue_time_99th_percentile	Response Queue Time spent in responding to LeaderAndIsr requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_queue_time_avg	Response Queue Time spent in responding to LeaderAndIsr requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_leader_and_isr_response_queue_time_max	Response Queue Time spent in responding to LeaderAndIsr requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_queue_time_median	Response Queue Time spent in responding to LeaderAndIsr requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_queue_time_min	Response Queue Time spent in responding to LeaderAndIsr requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_queue_time_rate	Response Queue Time spent in responding to LeaderAndIsr requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_queue_time_stddev	Response Queue Time spent in responding to LeaderAndIsr requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_send_time_75th_percentile	Response Send Time spent in responding to LeaderAndIsr requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_send_time_999th_percentile	Response Send Time spent in responding to LeaderAndIsr requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_send_time_99th_percentile	Response Send Time spent in responding to LeaderAndIsr requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_send_time_avg	Response Send Time spent in responding to LeaderAndIsr requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_send_time_max	Response Send Time spent in responding to LeaderAndIsr requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_send_time_median	Response Send Time spent in responding to LeaderAndIsr requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_send_time_min	Response Send Time spent in responding to LeaderAndIsr requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_send_time_rate	Response Send Time spent in responding to LeaderAndIsr requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_response_send_time_stddev	Response Send Time spent in responding to LeaderAndIsr requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_total_time_75th_percentile	Total Time spent in responding to LeaderAndIsr requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_leader_and_isr_total_time_999th_percentile	Total Time spent in responding to LeaderAndIsr requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_total_time_99th_percentile	Total Time spent in responding to LeaderAndIsr requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_total_time_avg	Total Time spent in responding to LeaderAndIsr requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_total_time_max	Total Time spent in responding to LeaderAndIsr requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_total_time_median	Total Time spent in responding to LeaderAndIsr requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_total_time_min	Total Time spent in responding to LeaderAndIsr requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_total_time_rate	Total Time spent in responding to LeaderAndIsr requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_and_isr_total_time_stddev	Total Time spent in responding to LeaderAndIsr requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_election_15min_rate	Leader elections: 15 Min Rate	message.units. elections per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_election_1min_rate	Leader elections: 1 Min Rate	message.units. elections per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_election_5min_rate	Leader elections: 5 Min Rate	message.units. elections per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_election_75th_percentile	Leader elections: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_election_999th_percentile	Leader elections: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_election_99th_percentile	Leader elections: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_election_avg	Leader elections: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_election_max	Leader elections: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_election_median	Leader elections: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_leader_election_min	Leader elections: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_election_rate	Leader elections: Samples	message.units. elections per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_election_stddev	Leader elections: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_leader_replicas	Number of leader replicas on broker	replicas	cluster, kafka, rack	CDH 5, CDH 6
kafka_log_flush_15min_rate	Rate of flushing Kafka logs to disk: 15 Min Rate	message.units. flushes per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_log_flush_1min_rate	Rate of flushing Kafka logs to disk: 1 Min Rate	message.units. flushes per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_log_flush_5min_rate	Rate of flushing Kafka logs to disk: 5 Min Rate	message.units. flushes per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_log_flush_75th_percentile	Rate of flushing Kafka logs to disk: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_log_flush_999th_percentile	Rate of flushing Kafka logs to disk: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_log_flush_99th_percentile	Rate of flushing Kafka logs to disk: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_log_flush_avg	Rate of flushing Kafka logs to disk: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_log_flush_max	Rate of flushing Kafka logs to disk: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_log_flush_median	Rate of flushing Kafka logs to disk: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_log_flush_min	Rate of flushing Kafka logs to disk: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_log_flush_rate	Rate of flushing Kafka logs to disk: Samples	message.units. flushes per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_log_flush_stddev	Rate of flushing Kafka logs to disk: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_max_replication_lag	Maximum replication lag on broker, across all fetchers, topics and partitions	messages	cluster, kafka, rack	CDH 5, CDH 6
kafka_memory_heap_committed	JVM heap committed memory	bytes	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_memory_heap_init	JVM heap initial memory	bytes	cluster, kafka, rack	CDH 5, CDH 6
kafka_memory_heap_max	JVM heap max used memory	bytes	cluster, kafka, rack	CDH 5, CDH 6
kafka_memory_heap_used	JVM heap used memory	bytes	cluster, kafka, rack	CDH 5, CDH 6
kafka_memory_total_committed	JVM heap and non-heap committed memory	bytes	cluster, kafka, rack	CDH 5, CDH 6
kafka_memory_total_init	JVM heap and non-heap initial memory	bytes	cluster, kafka, rack	CDH 5, CDH 6
kafka_memory_total_max	JVM heap and non-heap max initial memory	bytes	cluster, kafka, rack	CDH 5, CDH 6
kafka_memory_total_used	JVM heap and non-heap used memory	bytes	cluster, kafka, rack	CDH 5, CDH 6
kafka_messages_received_15min_rate	Number of messages written to topic on this broker: 15 Min Rate	messages per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_messages_received_1min_rate	Number of messages written to topic on this broker: 1 Min Rate	messages per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_messages_received_5min_rate	Number of messages written to topic on this broker: 5 Min Rate	messages per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_messages_received_avg_rate	Number of messages written to topic on this broker: Avg Rate	messages per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_messages_received_rate	Number of messages written to topic on this broker	messages per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_local_time_75th_percentile	Local Time spent in responding to Metadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_local_time_999th_percentile	Local Time spent in responding to Metadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_local_time_99th_percentile	Local Time spent in responding to Metadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_local_time_avg	Local Time spent in responding to Metadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_local_time_max	Local Time spent in responding to Metadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_local_time_median	Local Time spent in responding to Metadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_metadata_local_time_min	Local Time spent in responding to Metadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_local_time_rate	Local Time spent in responding to Metadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_local_time_stddev	Local Time spent in responding to Metadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_remote_time_75th_percentile	Remote Time spent in responding to Metadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_remote_time_999th_percentile	Remote Time spent in responding to Metadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_remote_time_99th_percentile	Remote Time spent in responding to Metadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_remote_time_avg	Remote Time spent in responding to Metadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_remote_time_max	Remote Time spent in responding to Metadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_remote_time_median	Remote Time spent in responding to Metadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_remote_time_min	Remote Time spent in responding to Metadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_remote_time_rate	Remote Time spent in responding to Metadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_remote_time_stddev	Remote Time spent in responding to Metadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_request_queue_time_75th_percentile	Request Queue Time spent in responding to Metadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_request_queue_time_999th_percentile	Request Queue Time spent in responding to Metadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_request_queue_time_99th_percentile	Request Queue Time spent in responding to Metadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_request_queue_time_avg	Request Queue Time spent in responding to Metadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_metadata_request_queue_time_max	Request Queue Time spent in responding to Metadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_request_queue_time_median	Request Queue Time spent in responding to Metadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_request_queue_time_min	Request Queue Time spent in responding to Metadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_request_queue_time_rate	Request Queue Time spent in responding to Metadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_request_queue_time_stddev	Request Queue Time spent in responding to Metadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_requests_15min_rate	Number of Metadata requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_requests_1min_rate	Number of Metadata requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_requests_5min_rate	Number of Metadata requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_requests_avg_rate	Number of Metadata requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_requests_rate	Number of Metadata requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_queue_time_75th_percentile	Response Queue Time spent in responding to Metadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_queue_time_999th_percentile	Response Queue Time spent in responding to Metadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_queue_time_99th_percentile	Response Queue Time spent in responding to Metadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_queue_time_avg	Response Queue Time spent in responding to Metadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_queue_time_max	Response Queue Time spent in responding to Metadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_metadata_response_queue_time_median	Response Queue Time spent in responding to Metadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_queue_time_min	Response Queue Time spent in responding to Metadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_queue_time_rate	Response Queue Time spent in responding to Metadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_queue_time_stddev	Response Queue Time spent in responding to Metadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_send_time_75th_percentile	Response Send Time spent in responding to Metadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_send_time_999th_percentile	Response Send Time spent in responding to Metadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_send_time_99th_percentile	Response Send Time spent in responding to Metadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_send_time_avg	Response Send Time spent in responding to Metadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_send_time_max	Response Send Time spent in responding to Metadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_send_time_median	Response Send Time spent in responding to Metadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_send_time_min	Response Send Time spent in responding to Metadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_send_time_rate	Response Send Time spent in responding to Metadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_response_send_time_stddev	Response Send Time spent in responding to Metadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_total_time_75th_percentile	Total Time spent in responding to Metadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_total_time_999th_percentile	Total Time spent in responding to Metadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_metadata_total_time_99th_percentile	Total Time spent in responding to Metadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_total_time_avg	Total Time spent in responding to Metadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_total_time_max	Total Time spent in responding to Metadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_total_time_median	Total Time spent in responding to Metadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_total_time_min	Total Time spent in responding to Metadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_total_time_rate	Total Time spent in responding to Metadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_metadata_total_time_stddev	Total Time spent in responding to Metadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_min_replication_rate	Minimum replication rate, across all fetchers, topics and partitions. Measured in average fetch requests per sec in the last minute	message.units.fetch_requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_network_processor_avg_idle_15min_rate	The average free capacity of the network processors: 15 Min Rate	message.units.percent_idle per message.units.singular.nanoseconds	cluster, kafka, rack	CDH 5, CDH 6
kafka_network_processor_avg_idle_1min_rate	The average free capacity of the network processors: 1 Min Rate	message.units.percent_idle per message.units.singular.nanoseconds	cluster, kafka, rack	CDH 5, CDH 6
kafka_network_processor_avg_idle_5min_rate	The average free capacity of the network processors: 5 Min Rate	message.units.percent_idle per message.units.singular.nanoseconds	cluster, kafka, rack	CDH 5, CDH 6
kafka_network_processor_avg_idle_avg_rate	The average free capacity of the network processors: Avg Rate	message.units.percent_idle per message.units.singular.nanoseconds	cluster, kafka, rack	CDH 5, CDH 6
kafka_network_processor_avg_idle_rate	The average free capacity of the network processors	message.units.percent_idle per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offline_partitions	Number of unavailable partitions	partitions	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_offset_commit_local_time_75th_percentile	Local Time spent in responding to OffsetCommit requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_local_time_999th_percentile	Local Time spent in responding to OffsetCommit requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_local_time_99th_percentile	Local Time spent in responding to OffsetCommit requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_local_time_avg	Local Time spent in responding to OffsetCommit requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_local_time_max	Local Time spent in responding to OffsetCommit requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_local_time_median	Local Time spent in responding to OffsetCommit requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_local_time_min	Local Time spent in responding to OffsetCommit requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_local_time_rate	Local Time spent in responding to OffsetCommit requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_local_time_stddev	Local Time spent in responding to OffsetCommit requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_remote_time_75th_percentile	Remote Time spent in responding to OffsetCommit requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_remote_time_999th_percentile	Remote Time spent in responding to OffsetCommit requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_remote_time_99th_percentile	Remote Time spent in responding to OffsetCommit requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_remote_time_avg	Remote Time spent in responding to OffsetCommit requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_remote_time_max	Remote Time spent in responding to OffsetCommit requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_remote_time_median	Remote Time spent in responding to OffsetCommit requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_remote_time_min	Remote Time spent in responding to OffsetCommit requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_offset_commit_remote_time_rate	Remote Time spent in responding to OffsetCommit requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_remote_time_stddev	Remote Time spent in responding to OffsetCommit requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_request_queue_time_75th_percentile	Request Queue Time spent in responding to OffsetCommit requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_request_queue_time_999th_percentile	Request Queue Time spent in responding to OffsetCommit requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_request_queue_time_99th_percentile	Request Queue Time spent in responding to OffsetCommit requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_request_queue_time_avg	Request Queue Time spent in responding to OffsetCommit requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_request_queue_time_max	Request Queue Time spent in responding to OffsetCommit requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_request_queue_time_median	Request Queue Time spent in responding to OffsetCommit requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_request_queue_time_min	Request Queue Time spent in responding to OffsetCommit requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_request_queue_time_rate	Request Queue Time spent in responding to OffsetCommit requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_request_queue_time_stddev	Request Queue Time spent in responding to OffsetCommit requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_requests_15min_rate	Number of OffsetCommit requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_requests_1min_rate	Number of OffsetCommit requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_requests_5min_rate	Number of OffsetCommit requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_requests_avg_rate	Number of OffsetCommit requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_offset_commit_requests_rate	Number of OffsetCommit requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_queue_time_75th_percentile	Response Queue Time spent in responding to OffsetCommit requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_queue_time_999th_percentile	Response Queue Time spent in responding to OffsetCommit requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_queue_time_99th_percentile	Response Queue Time spent in responding to OffsetCommit requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_queue_time_avg	Response Queue Time spent in responding to OffsetCommit requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_queue_time_max	Response Queue Time spent in responding to OffsetCommit requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_queue_time_median	Response Queue Time spent in responding to OffsetCommit requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_queue_time_min	Response Queue Time spent in responding to OffsetCommit requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_queue_time_rate	Response Queue Time spent in responding to OffsetCommit requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_queue_time_stddev	Response Queue Time spent in responding to OffsetCommit requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_send_time_75th_percentile	Response Send Time spent in responding to OffsetCommit requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_send_time_999th_percentile	Response Send Time spent in responding to OffsetCommit requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_send_time_99th_percentile	Response Send Time spent in responding to OffsetCommit requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_send_time_avg	Response Send Time spent in responding to OffsetCommit requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_send_time_max	Response Send Time spent in responding to OffsetCommit requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_offset_commit_response_send_time_median	Response Send Time spent in responding to OffsetCommit requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_send_time_min	Response Send Time spent in responding to OffsetCommit requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_send_time_rate	Response Send Time spent in responding to OffsetCommit requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_response_send_time_stddev	Response Send Time spent in responding to OffsetCommit requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_total_time_75th_percentile	Total Time spent in responding to OffsetCommit requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_total_time_999th_percentile	Total Time spent in responding to OffsetCommit requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_total_time_99th_percentile	Total Time spent in responding to OffsetCommit requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_total_time_avg	Total Time spent in responding to OffsetCommit requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_total_time_max	Total Time spent in responding to OffsetCommit requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_total_time_median	Total Time spent in responding to OffsetCommit requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_total_time_min	Total Time spent in responding to OffsetCommit requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_total_time_rate	Total Time spent in responding to OffsetCommit requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_commit_total_time_stddev	Total Time spent in responding to OffsetCommit requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_local_time_75th_percentile	Local Time spent in responding to OffsetFetch requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_local_time_999th_percentile	Local Time spent in responding to OffsetFetch requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_local_time_99th_percentile	Local Time spent in responding to OffsetFetch requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_offset_fetch_local_time_avg	Local Time spent in responding to OffsetFetch requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_local_time_max	Local Time spent in responding to OffsetFetch requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_local_time_median	Local Time spent in responding to OffsetFetch requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_local_time_min	Local Time spent in responding to OffsetFetch requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_local_time_rate	Local Time spent in responding to OffsetFetch requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_local_time_stddev	Local Time spent in responding to OffsetFetch requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_remote_time_75th_percentile	Remote Time spent in responding to OffsetFetch requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_remote_time_999th_percentile	Remote Time spent in responding to OffsetFetch requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_remote_time_99th_percentile	Remote Time spent in responding to OffsetFetch requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_remote_time_avg	Remote Time spent in responding to OffsetFetch requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_remote_time_max	Remote Time spent in responding to OffsetFetch requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_remote_time_median	Remote Time spent in responding to OffsetFetch requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_remote_time_min	Remote Time spent in responding to OffsetFetch requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_remote_time_rate	Remote Time spent in responding to OffsetFetch requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_remote_time_stddev	Remote Time spent in responding to OffsetFetch requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_request_queue_time_75th_percentile	Request Queue Time spent in responding to OffsetFetch requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_offset_fetch_request_queue_time_999th_percentile	Request Queue Time spent in responding to OffsetFetch requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_request_queue_time_99th_percentile	Request Queue Time spent in responding to OffsetFetch requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_request_queue_time_avg	Request Queue Time spent in responding to OffsetFetch requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_request_queue_time_max	Request Queue Time spent in responding to OffsetFetch requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_request_queue_time_median	Request Queue Time spent in responding to OffsetFetch requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_request_queue_time_min	Request Queue Time spent in responding to OffsetFetch requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_request_queue_time_rate	Request Queue Time spent in responding to OffsetFetch requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_request_queue_time_stddev	Request Queue Time spent in responding to OffsetFetch requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_requests_15min_rate	Number of OffsetFetch requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_requests_1min_rate	Number of OffsetFetch requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_requests_5min_rate	Number of OffsetFetch requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_requests_avg_rate	Number of OffsetFetch requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_requests_rate	Number of OffsetFetch requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_queue_time_75th_percentile	Response Queue Time spent in responding to OffsetFetch requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_queue_time_999th_percentile	Response Queue Time spent in responding to OffsetFetch requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_offset_fetch_response_queue_time_99th_percentile	Response Queue Time spent in responding to OffsetFetch requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_queue_time_avg	Response Queue Time spent in responding to OffsetFetch requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_queue_time_max	Response Queue Time spent in responding to OffsetFetch requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_queue_time_median	Response Queue Time spent in responding to OffsetFetch requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_queue_time_min	Response Queue Time spent in responding to OffsetFetch requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_queue_time_rate	Response Queue Time spent in responding to OffsetFetch requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_queue_time_stddev	Response Queue Time spent in responding to OffsetFetch requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_send_time_75th_percentile	Response Send Time spent in responding to OffsetFetch requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_send_time_999th_percentile	Response Send Time spent in responding to OffsetFetch requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_send_time_99th_percentile	Response Send Time spent in responding to OffsetFetch requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_send_time_avg	Response Send Time spent in responding to OffsetFetch requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_send_time_max	Response Send Time spent in responding to OffsetFetch requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_send_time_median	Response Send Time spent in responding to OffsetFetch requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_send_time_min	Response Send Time spent in responding to OffsetFetch requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_response_send_time_rate	Response Send Time spent in responding to OffsetFetch requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_offset_fetch_response_send_time_stddev	Response Send Time spent in responding to OffsetFetch requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_total_time_75th_percentile	Total Time spent in responding to OffsetFetch requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_total_time_999th_percentile	Total Time spent in responding to OffsetFetch requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_total_time_99th_percentile	Total Time spent in responding to OffsetFetch requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_total_time_avg	Total Time spent in responding to OffsetFetch requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_total_time_max	Total Time spent in responding to OffsetFetch requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_total_time_median	Total Time spent in responding to OffsetFetch requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_total_time_min	Total Time spent in responding to OffsetFetch requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_total_time_rate	Total Time spent in responding to OffsetFetch requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offset_fetch_total_time_stddev	Total Time spent in responding to OffsetFetch requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets	The size of the offsets cache	groups	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_groups	The number of consumer groups in the offsets cache	groups	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_local_time_75th_percentile	Local Time spent in responding to Offsets requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_local_time_999th_percentile	Local Time spent in responding to Offsets requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_local_time_99th_percentile	Local Time spent in responding to Offsets requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_local_time_avg	Local Time spent in responding to Offsets requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_local_time_max	Local Time spent in responding to Offsets requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_offsets_local_time_median	Local Time spent in responding to Offsets requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_local_time_min	Local Time spent in responding to Offsets requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_local_time_rate	Local Time spent in responding to Offsets requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_local_time_stddev	Local Time spent in responding to Offsets requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_remote_time_75th_percentile	Remote Time spent in responding to Offsets requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_remote_time_999th_percentile	Remote Time spent in responding to Offsets requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_remote_time_99th_percentile	Remote Time spent in responding to Offsets requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_remote_time_avg	Remote Time spent in responding to Offsets requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_remote_time_max	Remote Time spent in responding to Offsets requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_remote_time_median	Remote Time spent in responding to Offsets requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_remote_time_min	Remote Time spent in responding to Offsets requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_remote_time_rate	Remote Time spent in responding to Offsets requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_remote_time_stddev	Remote Time spent in responding to Offsets requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_request_queue_time_75th_percentile	Request Queue Time spent in responding to Offsets requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_request_queue_time_999th_percentile	Request Queue Time spent in responding to Offsets requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_request_queue_time_99th_percentile	Request Queue Time spent in responding to Offsets requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_offsets_request_queue_time_avg	Request Queue Time spent in responding to Offsets requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_request_queue_time_max	Request Queue Time spent in responding to Offsets requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_request_queue_time_median	Request Queue Time spent in responding to Offsets requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_request_queue_time_min	Request Queue Time spent in responding to Offsets requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_request_queue_time_rate	Request Queue Time spent in responding to Offsets requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_request_queue_time_stddev	Request Queue Time spent in responding to Offsets requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_requests_15min_rate	Number of Offsets requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_requests_1min_rate	Number of Offsets requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_requests_5min_rate	Number of Offsets requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_requests_avg_rate	Number of Offsets requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_requests_rate	Number of Offsets requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_queue_time_75th_percentile	Response Queue Time spent in responding to Offsets requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_queue_time_999th_percentile	Response Queue Time spent in responding to Offsets requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_queue_time_99th_percentile	Response Queue Time spent in responding to Offsets requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_queue_time_avg	Response Queue Time spent in responding to Offsets requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_offsets_response_queue_time_max	Response Queue Time spent in responding to Offsets requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_queue_time_median	Response Queue Time spent in responding to Offsets requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_queue_time_min	Response Queue Time spent in responding to Offsets requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_queue_time_rate	Response Queue Time spent in responding to Offsets requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_queue_time_stddev	Response Queue Time spent in responding to Offsets requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_send_time_75th_percentile	Response Send Time spent in responding to Offsets requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_send_time_999th_percentile	Response Send Time spent in responding to Offsets requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_send_time_99th_percentile	Response Send Time spent in responding to Offsets requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_send_time_avg	Response Send Time spent in responding to Offsets requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_send_time_max	Response Send Time spent in responding to Offsets requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_send_time_median	Response Send Time spent in responding to Offsets requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_send_time_min	Response Send Time spent in responding to Offsets requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_send_time_rate	Response Send Time spent in responding to Offsets requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_response_send_time_stddev	Response Send Time spent in responding to Offsets requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_total_time_75th_percentile	Total Time spent in responding to Offsets requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_offsets_total_time_999th_percentile	Total Time spent in responding to Offsets requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_total_time_99th_percentile	Total Time spent in responding to Offsets requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_total_time_avg	Total Time spent in responding to Offsets requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_total_time_max	Total Time spent in responding to Offsets requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_total_time_median	Total Time spent in responding to Offsets requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_total_time_min	Total Time spent in responding to Offsets requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_total_time_rate	Total Time spent in responding to Offsets requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_offsets_total_time_stddev	Total Time spent in responding to Offsets requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_partitions	Number of partitions (lead or follower replicas) on broker	partitions	cluster, kafka, rack	CDH 5, CDH 6
kafka_preferred_replica_imbalance	Number of partitions where the lead replica is not the preferred replica	partitions	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_local_time_75th_percentile	Local Time spent in responding to Produce requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_local_time_999th_percentile	Local Time spent in responding to Produce requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_local_time_99th_percentile	Local Time spent in responding to Produce requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_local_time_avg	Local Time spent in responding to Produce requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_local_time_max	Local Time spent in responding to Produce requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_local_time_median	Local Time spent in responding to Produce requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_local_time_min	Local Time spent in responding to Produce requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_local_time_rate	Local Time spent in responding to Produce requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_produce_local_time_stddev	Local Time spent in responding to Produce requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_remote_time_75th_percentile	Remote Time spent in responding to Produce requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_remote_time_999th_percentile	Remote Time spent in responding to Produce requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_remote_time_99th_percentile	Remote Time spent in responding to Produce requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_remote_time_avg	Remote Time spent in responding to Produce requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_remote_time_max	Remote Time spent in responding to Produce requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_remote_time_median	Remote Time spent in responding to Produce requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_remote_time_min	Remote Time spent in responding to Produce requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_remote_time_rate	Remote Time spent in responding to Produce requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_remote_time_stddev	Remote Time spent in responding to Produce requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_request_queue_time_75th_percentile	Request Queue Time spent in responding to Produce requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_request_queue_time_999th_percentile	Request Queue Time spent in responding to Produce requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_request_queue_time_99th_percentile	Request Queue Time spent in responding to Produce requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_request_queue_time_avg	Request Queue Time spent in responding to Produce requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_request_queue_time_max	Request Queue Time spent in responding to Produce requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_produce_request_queue_time_median	Request Queue Time spent in responding to Produce requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_request_queue_time_min	Request Queue Time spent in responding to Produce requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_request_queue_time_rate	Request Queue Time spent in responding to Produce requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_request_queue_time_stddev	Request Queue Time spent in responding to Produce requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_requests_15min_rate	Number of Produce requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_requests_1min_rate	Number of Produce requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_requests_5min_rate	Number of Produce requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_requests_avg_rate	Number of Produce requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_requests_rate	Number of Produce requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_queue_time_75th_percentile	Response Queue Time spent in responding to Produce requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_queue_time_999th_percentile	Response Queue Time spent in responding to Produce requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_queue_time_99th_percentile	Response Queue Time spent in responding to Produce requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_queue_time_avg	Response Queue Time spent in responding to Produce requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_queue_time_max	Response Queue Time spent in responding to Produce requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_queue_time_median	Response Queue Time spent in responding to Produce requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_produce_response_queue_time_min	Response Queue Time spent in responding to Produce requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_queue_time_rate	Response Queue Time spent in responding to Produce requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_queue_time_stddev	Response Queue Time spent in responding to Produce requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_send_time_75th_percentile	Response Send Time spent in responding to Produce requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_send_time_999th_percentile	Response Send Time spent in responding to Produce requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_send_time_99th_percentile	Response Send Time spent in responding to Produce requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_send_time_avg	Response Send Time spent in responding to Produce requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_send_time_max	Response Send Time spent in responding to Produce requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_send_time_median	Response Send Time spent in responding to Produce requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_send_time_min	Response Send Time spent in responding to Produce requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_send_time_rate	Response Send Time spent in responding to Produce requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_response_send_time_stddev	Response Send Time spent in responding to Produce requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_total_time_75th_percentile	Total Time spent in responding to Produce requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_total_time_999th_percentile	Total Time spent in responding to Produce requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_total_time_99th_percentile	Total Time spent in responding to Produce requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_produce_total_time_avg	Total Time spent in responding to Produce requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_total_time_max	Total Time spent in responding to Produce requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_total_time_median	Total Time spent in responding to Produce requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_total_time_min	Total Time spent in responding to Produce requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_total_time_rate	Total Time spent in responding to Produce requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_produce_total_time_stddev	Total Time spent in responding to Produce requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_producer_expires_15min_rate	Number of expired delayed producer requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_producer_expires_1min_rate	Number of expired delayed producer requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_producer_expires_5min_rate	Number of expired delayed producer requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_producer_expires_avg_rate	Number of expired delayed producer requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_producer_expires_rate	Number of expired delayed producer requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_producer_purgatory_delayed_requests	Number of requests delayed in the producer purgatory	requests	cluster, kafka, rack	CDH 5, CDH 6
kafka_producer_purgatory_size	Requests waiting in the producer purgatory. This should be non-zero when acks = -1 is used in producers	requests	cluster, kafka, rack	CDH 5, CDH 6
kafka_rejected_message_batches_15min_rate	Number of message batches sent by producers that the broker rejected for this topic: 15 Min Rate	message.units.message_batches per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_rejected_message_batches_1min_rate	Number of message batches sent by producers that the broker rejected for this topic: 1 Min Rate	message.units.message_batches per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_rejected_message_batches_5min_rate	Number of message batches sent by producers that the broker rejected for this topic: 5 Min Rate	message.units.message_batches per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_rejected_message_batches_avg_rate	Number of message batches sent by producers that the broker rejected for this topic: Avg Rate	message.units.message_batches per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_rejected_message_batches_rate	Number of message batches sent by producers that the broker rejected for this topic	message.units.message_batches per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_request_handler_avg_idle_15min_rate	The average free capacity of the request handler: 15 Min Rate	message.units.percent_idle per message.units.singular.nanoseconds	cluster, kafka, rack	CDH 5, CDH 6
kafka_request_handler_avg_idle_1min_rate	The average free capacity of the request handler: 1 Min Rate	message.units.percent_idle per message.units.singular.nanoseconds	cluster, kafka, rack	CDH 5, CDH 6
kafka_request_handler_avg_idle_5min_rate	The average free capacity of the request handler: 5 Min Rate	message.units.percent_idle per message.units.singular.nanoseconds	cluster, kafka, rack	CDH 5, CDH 6
kafka_request_handler_avg_idle_avg_rate	The average free capacity of the request handler: Avg Rate	message.units.percent_idle per message.units.singular.nanoseconds	cluster, kafka, rack	CDH 5, CDH 6
kafka_request_handler_avg_idle_rate	The average free capacity of the request handler	message.units.percent_idle per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_request_queue_size	Request Queue Size	requests	cluster, kafka, rack	CDH 5, CDH 6
kafka_response_queue_size	Response Queue Size	message.units.responses	cluster, kafka, rack	CDH 5, CDH 6
kafka_responses_being_sent	The number of responses being sent by the network processors	message.units.responses	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_local_time_75th_percentile	Local Time spent in responding to StopReplica requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_stop_replica_local_time_999th_percentile	Local Time spent in responding to StopReplica requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_local_time_99th_percentile	Local Time spent in responding to StopReplica requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_local_time_avg	Local Time spent in responding to StopReplica requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_local_time_max	Local Time spent in responding to StopReplica requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_local_time_median	Local Time spent in responding to StopReplica requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_local_time_min	Local Time spent in responding to StopReplica requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_local_time_rate	Local Time spent in responding to StopReplica requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_local_time_stddev	Local Time spent in responding to StopReplica requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_remote_time_75th_percentile	Remote Time spent in responding to StopReplica requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_remote_time_999th_percentile	Remote Time spent in responding to StopReplica requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_remote_time_99th_percentile	Remote Time spent in responding to StopReplica requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_remote_time_avg	Remote Time spent in responding to StopReplica requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_remote_time_max	Remote Time spent in responding to StopReplica requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_remote_time_median	Remote Time spent in responding to StopReplica requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_remote_time_min	Remote Time spent in responding to StopReplica requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_remote_time_rate	Remote Time spent in responding to StopReplica requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_stop_replica_remote_time_stddev	Remote Time spent in responding to StopReplica requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_request_queue_time_75th_percentile	Request Queue Time spent in responding to StopReplica requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_request_queue_time_999th_percentile	Request Queue Time spent in responding to StopReplica requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_request_queue_time_99th_percentile	Request Queue Time spent in responding to StopReplica requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_request_queue_time_avg	Request Queue Time spent in responding to StopReplica requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_request_queue_time_max	Request Queue Time spent in responding to StopReplica requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_request_queue_time_median	Request Queue Time spent in responding to StopReplica requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_request_queue_time_min	Request Queue Time spent in responding to StopReplica requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_request_queue_time_rate	Request Queue Time spent in responding to StopReplica requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_request_queue_time_stddev	Request Queue Time spent in responding to StopReplica requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_requests_15min_rate	Number of StopReplica requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_requests_1min_rate	Number of StopReplica requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_requests_5min_rate	Number of StopReplica requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_requests_avg_rate	Number of StopReplica requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_requests_rate	Number of StopReplica requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_stop_replica_response_queue_time_75th_percentile	Response Queue Time spent in responding to StopReplica requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_queue_time_999th_percentile	Response Queue Time spent in responding to StopReplica requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_queue_time_99th_percentile	Response Queue Time spent in responding to StopReplica requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_queue_time_avg	Response Queue Time spent in responding to StopReplica requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_queue_time_max	Response Queue Time spent in responding to StopReplica requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_queue_time_median	Response Queue Time spent in responding to StopReplica requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_queue_time_min	Response Queue Time spent in responding to StopReplica requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_queue_time_rate	Response Queue Time spent in responding to StopReplica requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_queue_time_stddev	Response Queue Time spent in responding to StopReplica requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_send_time_75th_percentile	Response Send Time spent in responding to StopReplica requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_send_time_999th_percentile	Response Send Time spent in responding to StopReplica requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_send_time_99th_percentile	Response Send Time spent in responding to StopReplica requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_send_time_avg	Response Send Time spent in responding to StopReplica requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_send_time_max	Response Send Time spent in responding to StopReplica requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_send_time_median	Response Send Time spent in responding to StopReplica requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_stop_replica_response_send_time_min	Response Send Time spent in responding to StopReplica requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_send_time_rate	Response Send Time spent in responding to StopReplica requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_response_send_time_stddev	Response Send Time spent in responding to StopReplica requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_total_time_75th_percentile	Total Time spent in responding to StopReplica requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_total_time_999th_percentile	Total Time spent in responding to StopReplica requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_total_time_99th_percentile	Total Time spent in responding to StopReplica requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_total_time_avg	Total Time spent in responding to StopReplica requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_total_time_max	Total Time spent in responding to StopReplica requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_total_time_median	Total Time spent in responding to StopReplica requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_total_time_min	Total Time spent in responding to StopReplica requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_total_time_rate	Total Time spent in responding to StopReplica requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_stop_replica_total_time_stddev	Total Time spent in responding to StopReplica requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_thread_count	JVM daemon and non-daemon thread count	threads	cluster, kafka, rack	CDH 5, CDH 6
kafka_unclean_leader_elections_15min_rate	Unclean leader elections. Cloudera recommends disabling unclean leader elections, to avoid potential data loss, so this should be 0: 15 Min Rate	message.units. elections per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_unclean_leader_elections_1min_rate	Unclean leader elections. Cloudera recommends disabling unclean leader elections, to avoid potential data loss, so this should be 0: 1 Min Rate	message.units. elections per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_unclean_leader_elections_5min_rate	Unclean leader elections. Cloudera recommends disabling unclean leader elections, to avoid potential data loss, so this should be 0: 5 Min Rate	message.units. elections per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_unclean_leader_elections_avg_rate	Unclean leader elections. Cloudera recommends disabling unclean leader elections, to avoid potential data loss, so this should be 0: Avg Rate	message.units. elections per message.units. singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_unclean_leader_elections_rate	Unclean leader elections. Cloudera recommends disabling unclean leader elections, to avoid potential data loss, so this should be 0	message.units. elections per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_under_replicated_partitions	Number of partitions with unavailable replicas	partitions	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_local_time_75th_percentile	Local Time spent in responding to UpdateMetadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_local_time_999th_percentile	Local Time spent in responding to UpdateMetadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_local_time_99th_percentile	Local Time spent in responding to UpdateMetadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_local_time_avg	Local Time spent in responding to UpdateMetadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_local_time_max	Local Time spent in responding to UpdateMetadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_local_time_median	Local Time spent in responding to UpdateMetadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_local_time_min	Local Time spent in responding to UpdateMetadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_local_time_rate	Local Time spent in responding to UpdateMetadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_local_time_stddev	Local Time spent in responding to UpdateMetadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_remote_time_75th_percentile	Remote Time spent in responding to UpdateMetadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_update_metadata_remote_time_999th_percentile	Remote Time spent in responding to UpdateMetadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_remote_time_99th_percentile	Remote Time spent in responding to UpdateMetadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_remote_time_avg	Remote Time spent in responding to UpdateMetadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_remote_time_max	Remote Time spent in responding to UpdateMetadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_remote_time_median	Remote Time spent in responding to UpdateMetadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_remote_time_min	Remote Time spent in responding to UpdateMetadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_remote_time_rate	Remote Time spent in responding to UpdateMetadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_remote_time_stddev	Remote Time spent in responding to UpdateMetadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_request_queue_time_75th_percentile	Request Queue Time spent in responding to UpdateMetadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_request_queue_time_999th_percentile	Request Queue Time spent in responding to UpdateMetadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_request_queue_time_99th_percentile	Request Queue Time spent in responding to UpdateMetadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_request_queue_time_avg	Request Queue Time spent in responding to UpdateMetadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_request_queue_time_max	Request Queue Time spent in responding to UpdateMetadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_request_queue_time_median	Request Queue Time spent in responding to UpdateMetadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_request_queue_time_min	Request Queue Time spent in responding to UpdateMetadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_update_metadata_request_queue_time_rate	Request Queue Time spent in responding to UpdateMetadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_request_queue_time_stddev	Request Queue Time spent in responding to UpdateMetadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_requests_15min_rate	Number of UpdateMetadata requests: 15 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_requests_1min_rate	Number of UpdateMetadata requests: 1 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_requests_5min_rate	Number of UpdateMetadata requests: 5 Min Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_requests_avg_rate	Number of UpdateMetadata requests: Avg Rate	requests per message.units.singular.second	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_requests_rate	Number of UpdateMetadata requests	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_queue_time_75th_percentile	Response Queue Time spent in responding to UpdateMetadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_queue_time_999th_percentile	Response Queue Time spent in responding to UpdateMetadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_queue_time_99th_percentile	Response Queue Time spent in responding to UpdateMetadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_queue_time_avg	Response Queue Time spent in responding to UpdateMetadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_queue_time_max	Response Queue Time spent in responding to UpdateMetadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_queue_time_median	Response Queue Time spent in responding to UpdateMetadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_queue_time_min	Response Queue Time spent in responding to UpdateMetadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_queue_time_rate	Response Queue Time spent in responding to UpdateMetadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_update_metadata_response_queue_time_stddev	Response Queue Time spent in responding to UpdateMetadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_send_time_75th_percentile	Response Send Time spent in responding to UpdateMetadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_send_time_999th_percentile	Response Send Time spent in responding to UpdateMetadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_send_time_99th_percentile	Response Send Time spent in responding to UpdateMetadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_send_time_avg	Response Send Time spent in responding to UpdateMetadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_send_time_max	Response Send Time spent in responding to UpdateMetadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_send_time_median	Response Send Time spent in responding to UpdateMetadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_send_time_min	Response Send Time spent in responding to UpdateMetadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_send_time_rate	Response Send Time spent in responding to UpdateMetadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_response_send_time_stddev	Response Send Time spent in responding to UpdateMetadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_total_time_75th_percentile	Total Time spent in responding to UpdateMetadata requests: 75th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_total_time_999th_percentile	Total Time spent in responding to UpdateMetadata requests: 999th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_total_time_99th_percentile	Total Time spent in responding to UpdateMetadata requests: 99th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_total_time_avg	Total Time spent in responding to UpdateMetadata requests: Avg	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_total_time_max	Total Time spent in responding to UpdateMetadata requests: Max	ms	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_update_metadata_total_time_median	Total Time spent in responding to UpdateMetadata requests: 50th Percentile	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_total_time_min	Total Time spent in responding to UpdateMetadata requests: Min	ms	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_total_time_rate	Total Time spent in responding to UpdateMetadata requests: Samples	requests per second	cluster, kafka, rack	CDH 5, CDH 6
kafka_update_metadata_total_time_stddev	Total Time spent in responding to UpdateMetadata requests: Standard Deviation	ms	cluster, kafka, rack	CDH 5, CDH 6
mem_rss	Resident memory used	bytes	cluster, kafka, rack	CDH 5, CDH 6
mem_swap	Amount of swap memory used by this role's process.	bytes	cluster, kafka, rack	CDH 5, CDH 6
mem_virtual	Virtual memory used	bytes	cluster, kafka, rack	CDH 5, CDH 6
oom_exits_rate	The number of times the role's backing process was killed due to an OutOfMemory error. This counter is only incremented if the Cloudera Manager "Kill When Out of Memory" option is enabled.	exits per second	cluster, kafka, rack	CDH 5, CDH 6
read_bytes_rate	The number of bytes read from the device	bytes per second	cluster, kafka, rack	CDH 5, CDH 6
unexpected_exits_rate	The number of times the role's backing process exited unexpectedly.	exits per second	cluster, kafka, rack	CDH 5, CDH 6
uptime	For a host, the amount of time since the host was booted. For a role, the uptime of the backing process.	seconds	cluster, kafka, rack	CDH 5, CDH 6
write_bytes_rate	The number of bytes written to the device	bytes per second	cluster, kafka, rack	CDH 5, CDH 6

Broker Topic Metrics

Metric Name	Description	Unit	Parents	CDH Version
kafka_bytes_fetched_15min_rate	Amount of data consumers fetched from this topic on this broker: 15 Min Rate	bytes per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_bytes_fetched_1min_rate	Amount of data consumers fetched from this topic on this broker: 1 Min Rate	bytes per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_bytes_fetched_5min_rate	Amount of data consumers fetched from this topic on this broker: 5 Min Rate	bytes per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_bytes_fetched_avg_rate	Amount of data consumers fetched from this topic on this broker: Avg Rate	bytes per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_bytes_fetched_rate	Amount of data consumers fetched from this topic on this broker	bytes per second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_bytes_received_15min_rate	Amount of data written to topic on this broker: 15 Min Rate	bytes per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_bytes_received_1min_rate	Amount of data written to topic on this broker: 1 Min Rate	bytes per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_bytes_received_5min_rate	Amount of data written to topic on this broker: 5 Min Rate	bytes per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_bytes_received_avg_rate	Amount of data written to topic on this broker: Avg Rate	bytes per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_bytes_received_rate	Amount of data written to topic on this broker	bytes per second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_bytes_rejected_15min_rate	Amount of data in messages rejected by broker for this topic: 15 Min Rate	bytes per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_bytes_rejected_1min_rate	Amount of data in messages rejected by broker for this topic: 1 Min Rate	bytes per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_bytes_rejected_5min_rate	Amount of data in messages rejected by broker for this topic: 5 Min Rate	bytes per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_bytes_rejected_avg_rate	Amount of data in messages rejected by broker for this topic: Avg Rate	bytes per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_bytes_rejected_rate	Amount of data in messages rejected by broker for this topic	bytes per second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_fetch_request_failures_15min_rate	Number of data read requests from consumers that brokers failed to process for this topic: 15 Min Rate	message. units. fetch_requests per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_fetch_request_failures_1min_rate	Number of data read requests from consumers that brokers failed to process for this topic: 1 Min Rate	message. units. fetch_requests per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_fetch_request_failures_5min_rate	Number of data read requests from consumers that brokers failed to process for this topic: 5 Min Rate	message. units. fetch_requests per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_fetch_request_failures_avg_rate	Number of data read requests from consumers that brokers failed to process for this topic: Avg Rate	message. units. fetch_requests per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_fetch_request_failures_rate	Number of data read requests from consumers that brokers failed to process for this topic	message. units. fetch_requests per second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
kafka_messages_received_15min_rate	Number of messages written to topic on this broker: 15 Min Rate	messages per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_messages_received_1min_rate	Number of messages written to topic on this broker: 1 Min Rate	messages per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_messages_received_5min_rate	Number of messages written to topic on this broker: 5 Min Rate	messages per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_messages_received_avg_rate	Number of messages written to topic on this broker: Avg Rate	messages per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_messages_received_rate	Number of messages written to topic on this broker	messages per second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_rejected_message_batches_15min_rate	Number of message batches sent by producers that the broker rejected for this topic: 15 Min Rate	message. units. message_batches per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_rejected_message_batches_1min_rate	Number of message batches sent by producers that the broker rejected for this topic: 1 Min Rate	message. units. message_batches per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_rejected_message_batches_5min_rate	Number of message batches sent by producers that the broker rejected for this topic: 5 Min Rate	message. units. message_batches per message. units. singular. second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6
kafka_rejected_message_batches_avg_rate	Number of message batches sent by producers that the broker rejected for this topic: Avg Rate	message. units. message_batches per message. units.	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
		singular. second		
kafka_rejected_message_batches_rate	Number of message batches sent by producers that the broker rejected for this topic	message. units. message_batches per second	cluster, kafka, kafka-kafka_broker, kafka_topic, rack	CDH 5, CDH 6

Mirror Maker Metrics

Metric Name	Description	Unit	Parents	CDH Version
alerts_rate	The number of alerts.	events per second	cluster, kafka, rack	CDH 5, CDH 6
cgroup_cpu_system_rate	CPU usage of the role's cgroup	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
cgroup_cpu_user_rate	User Space CPU usage of the role's cgroup	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
cgroup_mem_page_cache	Page cache usage of the role's cgroup	bytes	cluster, kafka, rack	CDH 5, CDH 6
cgroup_mem_rss	Resident memory of the role's cgroup	bytes	cluster, kafka, rack	CDH 5, CDH 6
cgroup_mem_swap	Swap usage of the role's cgroup	bytes	cluster, kafka, rack	CDH 5, CDH 6
cgroup_read_bytes_rate	Bytes read from all disks by the role's cgroup	bytes per second	cluster, kafka, rack	CDH 5, CDH 6
cgroup_read_ios_rate	Number of read I/O operations from all disks by the role's cgroup	ios per second	cluster, kafka, rack	CDH 5, CDH 6
cgroup_write_bytes_rate	Bytes written to all disks by the role's cgroup	bytes per second	cluster, kafka, rack	CDH 5, CDH 6
cgroup_write_ios_rate	Number of write I/O operations to all disks by the role's cgroup	ios per second	cluster, kafka, rack	CDH 5, CDH 6
cpu_system_rate	Total System CPU	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
cpu_user_rate	Total CPU user time	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
events_critical_rate	The number of critical events.	events per second	cluster, kafka, rack	CDH 5, CDH 6
events_important_rate	The number of important events.	events per second	cluster, kafka, rack	CDH 5, CDH 6
events_informational_rate	The number of informational events.	events per second	cluster, kafka, rack	CDH 5, CDH 6
fd_max	Maximum number of file descriptors	file descriptors	cluster, kafka, rack	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
fd_open	Open file descriptors.	file descriptors	cluster, kafka, rack	CDH 5, CDH 6
health_bad_rate	Percentage of Time with Bad Health	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
health_concerning_rate	Percentage of Time with Concerning Health	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
health_disabled_rate	Percentage of Time with Disabled Health	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
health_good_rate	Percentage of Time with Good Health	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
health_unknown_rate	Percentage of Time with Unknown Health	seconds per second	cluster, kafka, rack	CDH 5, CDH 6
mem_rss	Resident memory used	bytes	cluster, kafka, rack	CDH 5, CDH 6
mem_swap	Amount of swap memory used by this role's process.	bytes	cluster, kafka, rack	CDH 5, CDH 6
mem_virtual	Virtual memory used	bytes	cluster, kafka, rack	CDH 5, CDH 6
oom_exits_rate	The number of times the role's backing process was killed due to an OutOfMemory error. This counter is only incremented if the Cloudera Manager "Kill When Out of Memory" option is enabled.	exits per second	cluster, kafka, rack	CDH 5, CDH 6
read_bytes_rate	The number of bytes read from the device	bytes per second	cluster, kafka, rack	CDH 5, CDH 6
unexpected_exits_rate	The number of times the role's backing process exited unexpectedly.	exits per second	cluster, kafka, rack	CDH 5, CDH 6
uptime	For a host, the amount of time since the host was booted. For a role, the uptime of the backing process.	seconds	cluster, kafka, rack	CDH 5, CDH 6
write_bytes_rate	The number of bytes written to the device	bytes per second	cluster, kafka, rack	CDH 5, CDH 6

Replica Metrics

Metric Name	Description	Unit	Parents	CDH Version
kafka_log_end_offset	The offset of the next message that will be appended to the log	message. units.offset	cluster, kafka, kafka-kafka_broker, kafka_broker_topic, kafka_topic, rack	CDH 5, CDH 6
kafka_log_start_offset	The earliest message offset in the log	message. units.offset	cluster, kafka, kafka-kafka_broker,	CDH 5, CDH 6

Metric Name	Description	Unit	Parents	CDH Version
			kafka_broker_topic, kafka_topic, rack	
kafka_num_log_segments	The number of segments in the log	message. units. segments	cluster, kafka, kafka-kafka_broker, kafka_broker_topic, kafka_topic, rack	CDH 5, CDH 6
kafka_size	The size of the log	bytes	cluster, kafka, kafka-kafka_broker, kafka_broker_topic, kafka_topic, rack	CDH 5, CDH 6

Useful Shell Command Reference

Hardware Information

There are many ways to get information about hardware:

CPU	`cat /proc/cpuinfo` `lscpu`

Memory	`cat /proc/meminfo`
	`vmstat -s`
	`free -m or free -g`

Network interface	`ip link show` `netstat -i`
I/O device (hard drive)	`lsblk -d` `fdisk -l`
Virtual environment	`virt-what`

Disk Space

`df` or `mount` show the disks mounted and can be used to show disk space.

On a file or directory level, the du command is useful for seeing how much disk space is being used.

I/O Activity and Utilization

`iostat` and `sar` come with Linux package sysstat-9.0.4-27. `iostat` is used for tracking I/O performance. The recommended options are:

- `-d` for disk utilization
- `-m` for calculations in MB/sec
- `-x` for extended report
- `-t` *sec* to repeat statistics every *sec* seconds

`sar` has several forms of output:

- `-b` for I/O and transfer rate statistics
- `-d` for block device activity

- `-n` for network statistics
- `-v` for various file system statistics

File Descriptor Usage

`lsof` is used to identify mapping between processes and open files. By passing multiple arguments, `lsof` can be used to help isolate the output:

- `lsof <filename>` to list processes that have `<filename>` open.
- `lsof -p <pid>` to list all files opened by the process corresponding to `<pid>`.
- `lsof -r <secs>` to keep producing output with a period of `<secs>`.

Network Ports, States, and Connections

- `nc` (`netcat`) or `ss` (socket statistics) are good for showing network activity.
- `netstat` is the Swiss-army knife tool for network interfaces.
- `tcpdump` or Wireshark with Kafka filter should be good for packet sniffing.

Process Information

- `top` shows a sorted list of processes.
- `ps` shows a snapshot list of processes. Arguments can be used to filter the output.
- `ps -o min_flt,maj_flt` *`pid`* shows page fault information.

Kernel Configuration

- `ulimit -a` is used to display kernel limits and shows which flags affect which kernel settings.
- `ulimit -n` *`FD`* to set a limit on open file descriptors.

Kafka Public APIs

What is a Public API

The following parts of Apache Kafka in CDH are considered as public APIs:

- Kafka wire protocol format: the format itself might change, but brokers will be able to use the old format as long as documentation and upgrade instructions are followed properly.
- Binary log format: the format itself might change, but brokers will be able to use the old format as long as documentation and upgrade instructions are followed properly.
- Interfaces and classes in the following packages:
  - org/apache/kafka/common/serialization
  - org/apache/kafka/common/errors
  - org/apache/kafka/clients/producer
  - org/apache/kafka/clients/consumer
- Command-line admin tools: arguments, except ZooKeeper related options, that are subject to change and/or removal.
- `HttpMetricsReporter`: existing fields will stay backward compatible, but new fields may be introduced. The only public API of `HttpMetricsReporter` is the /api/metrics REST endpoint. For a list of supported metrics, see Kafka Metrics.
- Properties, excluding their default values
- Config file content and format, and the effect of configuration attributes
- Endpoints

What is NOT a public API

There are structures that third parties might regard as an interface but Cloudera Kafka distributions do not consider them public APIs. In general, any API that is not listed as public in the What is a Public API section should be considered private, and client code should not rely on behavior/data content or format. Some examples are:

- Data structures in ZooKeeper: the content and format what Kafka stores in ZooKeeper are internal implementation details.
- Authorizer interface: the only supported authorizer in CHD is the Sentry one.
- AdminClient: it is a new and rapidly evolving part of Kafka, so Cloudera can't provide the same guarantees as for other interfaces.
- Interfaces marked with the `@Evolving` or `@Unstable` annotations in the Kafka source code
- Index files generated by Kafka
- Application log file content and format (for example what Log4J/SLF4J/… produces)
- Any classes used for testing
- Relying on transitive dependencies: any dependency pulled in by Kafka
- Any other interfaces not listed above
- Anything that Cloudera does not support, even if it fits the definition of a public API

Kafka Frequently Asked Questions

This is intended to be an easy to understand FAQ on the topic of Kafka. One part is for beginners, one for advanced users and use cases. We hope you find it fruitful. If you are missing a question, please send it to your favorite Cloudera representative and we'll populate this FAQ over time.

Basics

What is Kafka?

Kafka is a streaming message platform. Breaking it down a bit further:

"Streaming": Lots of messages (think tens or hundreds of thousands) being sent frequently by publishers ("producers"). Message polling occurring frequently by lots of subscribers ("consumers").

"Message": From a technical standpoint, a key value pair. From a non-technical standpoint, a relatively small number of bytes (think hundreds to a few thousand bytes).

If this isn't your planned use case, Kafka *may not* be the solution you are looking for. Contact your favorite Cloudera representative to discuss and find out. It is better to understand what you can and cannot do upfront than to go ahead based on some enthusiastic arbitrary vendor message with a solution that will not meet your expectations in the end.

What is Kafka designed for?

Kafka was designed at LinkedIn to be a horizontally scaling publish-subscribe system. It offers a great deal of configurability at the system- and message-level to achieve these performance goals. There are well documented cases (Uber and LinkedIn) that showcase how well Kafka can scale when everything is done right.

What is Kafka not well fitted for (or what are the tradeoffs)?

It's very easy to get caught up in all the things that Kafka can be used for without considering the tradeoffs. Kafka configuration is also not automatic. You need to understand each of your use cases to determine which configuration properties can be used to tune (and retune!) Kafka for each use case.

Some more specific examples where you need to be deeply knowledgeable and careful when configuring are:

- **Using Kafka as your microservices communication hub**

  Kafka can replace both the message queue and the services discovery part of your software infrastructure. However, this is generally at the cost of some added latency as well as the need to monitor a new complex system (i.e. your Kafka cluster).

- **Using Kafka as long-term storage**

  While Kafka does have a way to configure message retention, it's primarily designed for low latency message delivery. Kafka does not have any support for the features that are usually associated with filesystems (such as metadata or backups). As such, using some form of long-term ingestion, such as HDFS, is recommended instead.

- **Using Kafka as an end-to-end solution**

  Kafka is only part of a solution. There are a lot of best practices to follow and support tools to build before you can get the most out of it (see this wise LinkedIn post).

- **Deploying Kafka without the right support**

  Uber has given some numbers for their engineering organization. These numbers could help give you an idea what it takes to reach that kind of scale: 1300 microservers, 2000 engineers.

Where can I get a general Kafka overview?

The first four sections (Introduction, Setup, Clients, Brokers) of the CDH 6 Kafka Documentation cover the basics and design of Kafka. This should serve as a good starting point. If you have any remaining questions after reading that documentation, come to this FAQ or talk to your favorite Cloudera representative about training or a best practices deep dive.

Where does Kafka fit well into an Analytic DB solution?

ADB deployments benefit from Kafka by utilizing it for data ingest. Data can then populate tables for various analytics workloads. For ad hoc BI the real-time aspect is less critical, but the ability to utilize the same data used in real time applications, in BI and analytics as well, is a benefit that Cloudera's platform provides, as you will have Kafka for both purposes, already integrated, secured, governed and centrally managed.

Where does Kafka fit well into an Operational DB solution?

Kafka is commonly used in the real-time, mission-critical world of Operational DB deployments. It is used to ingest data and allow immediate serving to other applications and services via Kudu or HBase. The benefit of utilizing Kafka in the Cloudera platform for ODB is the integration, security, governance and central management. You avoid the risks and costs of siloed architecture and "yet another solution" to support.

What is a Kafka consumer?

If Kafka is the system that stores messages, then a consumer is the part of your system that reads those messages from Kafka.

While Kafka does come with a command line tool that can act as a consumer, practically speaking, you will most likely write Java code using the KafkaConsumer API for your production system.

What is a Kafka producer?

While consumers read from a Kafka cluster, producers write to a Kafka cluster.

Similar to the consumer (see previous question), your producer is also custom Java code for your particular use case.

Your producer may need some tuning for write performance and SLA guarantees, but will generally be simpler (fewer error cases) to tune than your consumer.

What functionality can I call in my Kafka Java code?

The best way to get more information on what functionality you can call in your Kafka Java code is to look at the Java docs. And read very carefully!

What's a good size of a Kafka record if I care about performance and stability?

There is an older blog post from 2014 from LinkedIn titled: Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines). In the "Effect of Message Size" section, you can see two charts which indicate that Kafka throughput starts being affected at a record size of 100 bytes through 1000 bytes and bottoming out around 10000 bytes. In general, keeping topics specific and keeping message sizes deliberately small helps you get the most out of Kafka.

Excerpting from Deploying Apache Kafka: A Practical FAQ:

How to send large messages or payloads through Kafka?

Cloudera benchmarks indicate that Kafka reaches maximum throughput with message sizes of around 10 KB. Larger messages show decreased throughput. However, in certain cases, users need to send messages much larger than 10 KB.

If the message payload sizes are in the order of 100s of MB, consider exploring the following alternatives:

- If shared storage is available (HDFS, S3, NAS), place the large payload on shared storage and use Kafka just to send a message with the payload location.
- Handle large messages by chopping them into smaller parts before writing into Kafka, using a message key to make sure all the parts are written to the same partition so that they are consumed by the same Consumer, and re-assembling the large message from its parts when consuming.

Where can I get Kafka training?

You have many options. Cloudera provides training as listed in the next two questions. You can also ask your Resident Solution Architect to do a deep dive on Kafka architecture and best practices. And you could always engage in the community to get insight and expertise on specific topics.

Where can I get basic Kafka training?

Cloudera training offers a basic on-demand training for Kafka[1].

This covers basics of Kafka architecture, messages, ordering, and a few slides (code examples) of (to my knowledge) an older version of the Java API. It also covers using Flume + Kafka.

Where can I get Kafka developer training?

Kafka developer training is included in Cloudera's Developer Training for Apache Spark and Hadoop[2].

Use Cases

Like most Open Source projects, Kafka provides a lot of configuration options to maximize performance. In some cases, it is not obvious how best to map your specific use case to those configuration options. We attempt to address some of those situations below.

What can I do to ensure that I never lose a Kafka event?

This is a simple question which has lots of far-reaching implications for your entire Kafka setup. A complete answer includes the next few related FAQs and their answers.

What is the recommended node hardware for best reliability?

Operationally, you need to make sure your Kafka cluster meets the following hardware setup:

1. Have a 3 or 5 node cluster only running Zookeeper (higher only necessary at largest scales).
2. Have at least a 3 node cluster only running Kafka.
3. Have the disks on the Kafka cluster running in RAID 10. (Required for resiliency against disk failure.)
4. Have sufficient memory for both the Kafka and Zookeeper roles in the cluster. (Recommended: 4GB for the broker, the rest of memory automatically used by the kernel as file cache.)
5. Have sufficient disk space on the Kafka cluster.
6. Have a sufficient number of disks to handle the bandwidth requirements for Kafka and Zookeeper.
7. You need a number of nodes greater than or equal to the highest replication factor you expect to use.

What are the network requirements for best reliability?

Kafka expects a reliable, low-latency connection between the brokers and the Zookeeper nodes.

1. The number of network hops between the Kafka cluster and the Zookeeper cluster is relatively low.
2. Have highly reliable network services (such as DNS).

What are the system software requirements for best reliability?

Assuming you're following the recommendations of the previous two questions, the actual system outside of Kafka must be configured properly.

1. The kernel must be configured for maximum I/O usage that Kafka requires.
    a. Large page cache
    b. Maximum file descriptions
    c. Maximum file memory map limits
2. Kafka JVM configuration settings:
    a. Brokers generally don't need more than 4GB-8GB of heap space.
    b. Run with the +G1GC garbage collection using Java 8 or later.

How can I configure Kafka to ensure that events are stored reliably?

The following recommendations for Kafka configuration settings make it extremely difficult for data loss to occur.

1. Producer
    a. `block.on.buffer.full=true`
    b. `retries=Long.MAX_VALUE`
    c. `acks=all`
    d. `max.in.flight.requests.per.connections=1`
    e. Remember to close the producer when it is finished or when there is a long pause.
2. Broker
    a. `Topic replication.factor >= 3`
    b. `Min.insync.replicas = 2`
    c. Disable unclean leader election
3. Consumer
    a. Disable `enable.auto.commit`
    b. Commit offsets after messages are processed by your consumer client(s).

If you have more than 3 hosts, you can increase the broker settings appropriately on topics that need more protection against data loss.

Once I've followed all the previous recommendations, my cluster should never lose data, right?

Kafka **does not ensure** that data loss never occurs. There are the following tradeoffs:

1. Throughput vs. reliability. For example, the higher the replication factor, the more resilient your setup will be against data loss. However, to make those extra copies takes time and can affect throughput.
2. Reliability vs. free disk space. Extra copies due to replication use up disk space that would otherwise be used for storing events.

Beyond the above design tradeoffs, there are also the following issues:

- To ensure events are consumed you need to monitor your Kafka brokers and topics to verify sufficient consumption rates are sustained to meet your ingestion requirements.
- Ensure that replication is enabled on any topic that requires consumption guarantees. This protects against Kafka broker failure and host failure.
- Kafka is designed to store events for a defined duration after which the events are deleted. You can increase the duration that events are retained up to the amount of supporting storage space.
- You will always run out of disk space unless you add more nodes to the cluster.

My Kafka events must be processed in order. How can I accomplish this?

After your topic is configured with partitions, Kafka sends each record (based on key/value pair) to a particular partition based on key. So, any given key, the corresponding records are “in order” within a partition.

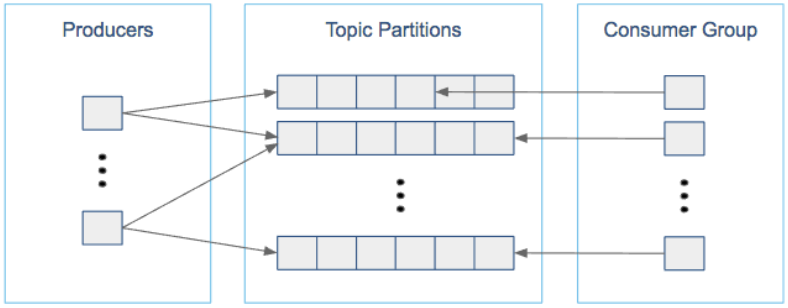
For global ordering, you have two options:

1. Your topic must consist of one partition (but a higher replication factor could be useful for redundancy and failover). However, this will result in very limited message throughput.
2. You configure your topic with a small number of partitions and perform the ordering after the consumer has pulled data. This does not result in **guaranteed** ordering, but, given a large enough time window, will likely be equivalent.

Conversely, it is best to take Kafka’s partitioning design into consideration when designing your Kafka setup rather than rely on global ordering of events.

How do I size my topic? *Alternatively: What is the “right” number of partitions for a topic?*

Choosing the proper number of partitions for a topic is the key to achieving a high degree of parallelism with respect to writes to and reads and to distribute load. Evenly distributed load over partitions is a key factor to have good throughput (avoid hot spots). Making a good decision requires estimation based on the desired throughput of producers and consumers per partition.



For example, if you want to be able to read 1 GB/sec, but your consumer is only able process 50 MB/sec, then you need at least 20 partitions and 20 consumers in the consumer group. Similarly, if you want to achieve the same for producers, and 1 producer can only write at 100 MB/sec, you need 10 partitions. In this case, if you have 20 partitions, you can maintain 1 GB/sec for producing and consuming messages. You should adjust the exact number of partitions to number of consumers or producers, so that each consumer and producer achieve their target throughput.

So a simple formula could be:

$$\#Partitions = max(N_P, N_C)$$

where:

- N_P is the number of required producers determined by calculating: T_T/T_P
- N_C is the number of required consumers determined by calculating: T_T/T_C
- T_T is the total expected throughput for our system
- T_P is the max throughput of a single producer to a single partition
- T_C is the max throughput of a single consumer from a single partition

This calculation gives you a rough indication of the number of partitions. It's a good place to start. Keep in mind the following considerations for improving the number of partitions after you have your system in place:

- The number of partitions can be specified at topic creation time or later.
- Increasing the number of partitions also affects the number of open file descriptors. So make sure you set file descriptor limit properly.
- Reassigning partitions can be very expensive, and therefore it's better to over- than under-provision.
- Changing the number of partitions that are based on keys is challenging and involves manual copying (see Kafka Administration on page 60).

- Reducing the number of partitions is not currently supported. Instead, create a new a topic with a lower number of partitions and copy over existing data.
- Metadata about partitions are stored in ZooKeeper in the form of `znodes`. Having a large number of partitions has effects on ZooKeeper and on client resources:
  - Unneeded partitions put extra pressure on ZooKeeper (more network requests), and might introduce delay in controller and/or partition leader election if a broker goes down.
  - Producer and consumer clients need more memory, because they need to keep track of more partitions and also buffer data for all partitions.
- As guideline for optimal performance, you should not have more than 4000 partitions per broker and not more than 200,000 partitions in a cluster.

Make sure consumers don't lag behind producers by monitoring consumer lag. To check consumers' position in a consumer group (that is, how far behind the end of the log they are), use the following command:

```
$ kafka-consumer-groups --bootstrap-server BROKER_ADDRESS --describe --group
CONSUMER_GROUP --new-consumer
```

How can I scale a topic that's already deployed in production?

Recall the following facts about Kafka:

- When you create a topic, you set the number of partitions. The higher the partition count, the better the parallelism and the better the events are spread somewhat evenly through the cluster.
- In most cases, as events go to the Kafka cluster, events with the same key go to the same partition. This is a consequence of using a hash function to determine which key goes to which partition.

Now, you might assume that scaling means increasing the number of partitions in a topic. However, due to the way hashing works, simply increasing the number of partitions means that you will lose the "events with the same key go to the same partition" fact.

Given that, there are two options:

1. Your cluster may not be scaling well because the partition loads are not balanced properly (for example, one broker has four very active partitions, while another has none). In those cases, you can use the `kafka-reassign-partitions` script to manually balance partitions.
2. Create a new topic with more partitions, pause the producers, copy data over from the old topic, and then move the producers and consumers over to the new topic. This can be a bit tricky operationally.

How do I rebalance my Kafka cluster?

This one comes up when new nodes or disks are added to existing nodes. Partitions are not automatically balanced. If a topic already has a number of nodes equal to the replication factor (typically 3), then adding disks does not help with rebalancing.

Using the `kafka-reassign-partitions` command after adding new hosts is the recommended method.

Caveats

There are several caveats to using this command:

- It is highly recommended that you minimize the volume of replica changes to make sure the cluster remains healthy. Say, instead of moving ten replicas with a single command, move two at a time.
- It is not possible to use this command to make an out-of-sync replica into the leader partition.
- If too many replicas are moved, then there could be serious performance impact on the cluster. When using the `kafka-reassign-partitions` command, look at the partition counts and sizes. From there, you can test various partition sizes along with the `--throttle` flag to determine what volume of data can be copied without affecting broker performance significantly.
- Given the earlier restrictions, it is best to use this command only when all brokers and topics are healthy.

How do I monitor my Kafka cluster?

As of Cloudera Enterprise 5.14, Cloudera Manager has monitoring for a Kafka cluster.

Currently, there are three GitHub projects as well that provide additional monitoring functionality:

- Doctor Kafka[3] (Pinterest, Apache 2.0 License)
- Kafka Manager[4] (Yahoo, Apache 2.0 License)
- Cruise Control[5] (LinkedIn, BSD 2-clause License)

These projects are Apache-compatible licensed, but are not Open Source (no community, bug filing, or transparency).

What are the best practices concerning consumer group.id?

The `group.id` is just a string that helps Kafka track which consumers are related (by having the same group id).

- In general, timestamps as part of `group.id` are not useful. Because each `group.id` corresponds to multiple consumers, you cannot have a unique timestamp for each consumer.
- Add any helpful identifiers. This could be related to a group (for example, transactions, marketing), purpose (fraud, alerts), or technology (Flume, Spark).

How do I monitor consumer group lag?

This is typically done using the `kafka-consumer-groups` command line tool. Copying directly from the upstream documentation[6], we have this example output (reformatted for readability):

```
$ bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group
 my-group
TOPIC  PARTITION CURRENT-OFFSET LOG-END-OFFSET LAG CONSUMER-ID       HOST
CLIENT-ID
my-topic   0           2             4         2  consumer-1-69d6 /127.0.0.1
consumer-1
my-topic   1           2             3         1  consumer-1-69d6 /127.0.0.1
consumer-1
my-topic   2           2             3         1  consumer-2-9bb2 /127.0.0.1
consumer-2
```

In general, if everything is going well with a particular topic, each consumer's `CURRENT-OFFSET` should be up-to-date or nearly up-to-date with the `LOG-END-OFFSET`. From this command, you can determine whether a particular host or a particular partition is having issues keeping up with the data rate.

How do I reset the consumer offset to an arbitrary value?

This is also done using the `kafka-consumer-groups` command line tool. This is generally an administration feature used to get around corrupted records, data loss, or recovering from failure of the broker or host. Aside from those special cases, using the command line tool for this purpose is not recommended.

By using the `--execute --reset-offsets` flags, you can change the consumer offsets for a consumer group (or even all groups) to a specific setting based on each partitions log's beginning/end or a fixed timestamp. Typing the `kafka-consumer-groups` command with no arguments will give you the complete help output.

How do I configure MirrorMaker for bi-directional replication across DCs?

Mirror Maker is a one way copy of one or more topics from a Source Kafka Cluster to a Destination Kafka Cluster. Given this restriction on Mirror Maker, you need to run two instances, one to copy from A to B and another to copy from B to A.

In addition, consider the following:

- Cloudera recommends using the "pull" model for Mirror Maker, meaning that the Mirror Maker instance that is writing to the destination is running on a host "near" the destination cluster.
- The topics must be unique across the two clusters being copied.

- On secure clusters, the source cluster and destination cluster must be in the same Kerberos realm.

How does the consumer max retries vs timeout work?

With the newer versions of Kafka, consumers have two ways they communicate with brokers.

- Retries: This is generally related to reading data. When a consumer reads from a brokers, it's possible for that attempt to fail due to problems such as intermittent network outages or I/O issues on the broker. To improve reliability, the consumer retries (up to the configured `max.retries` value) before actually failing to read a log offset.
- Timeout. This term is a bit vague because there are two timeouts related to consumers:
  - Poll Timeout: This is the timeout between calls to `KafkaConsumer.poll()`. This timeout is set based on whatever read latency requirements your particular use case needs.
  - Heartbeat Timeout: The newer consumer has a "heartbeat thread" which give a heartbeat to the broker (actually the Group Coordinator within a broker) to let the broker know that the consumer is still alive. This happens on a regular basis and if the broker doesn't receive at least one heartbeat within the timeout period, it assumes the consumer is dead and disconnects it.

How do I size my Kafka cluster?

There are several considerations for sizing your Kafka cluster.

- **Disk space**

  Disk space will primarily consist of your Kafka data and broker logs. When in debug mode, the broker logs can get quite large (10s to 100s of GB), so reserving a significant amount of space could save you some future headaches.

  For Kafka data, you need to perform estimates on message size, number of topics, and redundancy. Also remember that you will be using RAID10 for Kafka's data, so half your hard drives will go towards redundancy. From there, you can calculate how many drives will be needed.

  In general, you will want to have more hosts than the minimum suggested by the number of drives. This leaves room for growth and some scalability headroom.

- **Zookeeper nodes**

  One node is fine for a test cluster. Three is standard for most Kafka clusters. At large scale, five nodes is fairly common for reliability.

- **Looking at leader partition count/bandwidth usage**

  This is likely the metric with the highest variability. Any Kafka broker will be overloaded if it has too many leader partitions. In the worst cases, each leader partition requires high bandwidth, high message rates, or both. For other topics, leader partitions will be a tiny fraction of what a broker can handle (limited by software and hardware). To estimate an average that works on a per-host basis, try grouping topics by partition data throughput requirements, such as 2 high bandwidth data partitions, 4 medium bandwidth data partitions, 20 small bandwidth data partitions. From there, you can determine how many hosts are needed.

How can I combine Kafka with Flume to ingest into HDFS?

We have two blog posts on using Kafka with Flume:

- The original post: Flafka: Apache Flume Meets Apache Kafka for Event Processing
- This updated version for CDH 5.8/Apache Kafka 0.9/Apache Flume 1.7: New in Cloudera Enterprise 5.8: Flafka Improvements for Real-Time Data Ingest

How can I build a Spark streaming application that consumes data from Kafka?

You will need to set up your development environment to use both Spark libraries and Kafka libraries:

- Building Spark Applications

- The kafka-examples directory on Cloudera's public GitHub has an example `pom.xml`.

From there, you should be able to read data using the KafkaConsumer class and using Spark libraries for real-time data processing. The blog post Reading data securely from Apache Kafka to Apache Spark has a pointer to a GitHub repository that contains a word count example.

For further background, read the blog post Architectural Patterns for Near Real-Time Data Processing with Apache Hadoop.

References

1. Kafka basic training: https://ondemand.cloudera.com/courses/course-v1:Cloudera+Kafka+201601/info
2. Kafka developer training: https://ondemand.cloudera.com/courses/course-v1:Cloudera+DevSH+201709/info
3. Doctor Kafka: http://github.com/pinterest/doctorkafka
4. Kafka manager: http://github.com/yahoo/kafka-manager
5. Cruise control: http://github.com/linkedin/cruise-control
6. Upstream documentation: http://kafka.apache.org/documentation/#basic_ops_consumer_lag

Appendix: Apache License, Version 2.0

SPDX short identifier: Apache-2.0

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims

licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and
2. You must cause any modified files to carry prominent notices stating that You changed the files; and
3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

```
Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```