cloudera®

Cloudera Distribution of Kafka

Important Notice

The information in this document is subject to change without notice. Cloudera shall not be liable for any damages resulting from technical errors or omissions which may be present in this document, or from use of this document.

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: Cloudera Distribution of Apache Kafka
Date: July 20, 2021

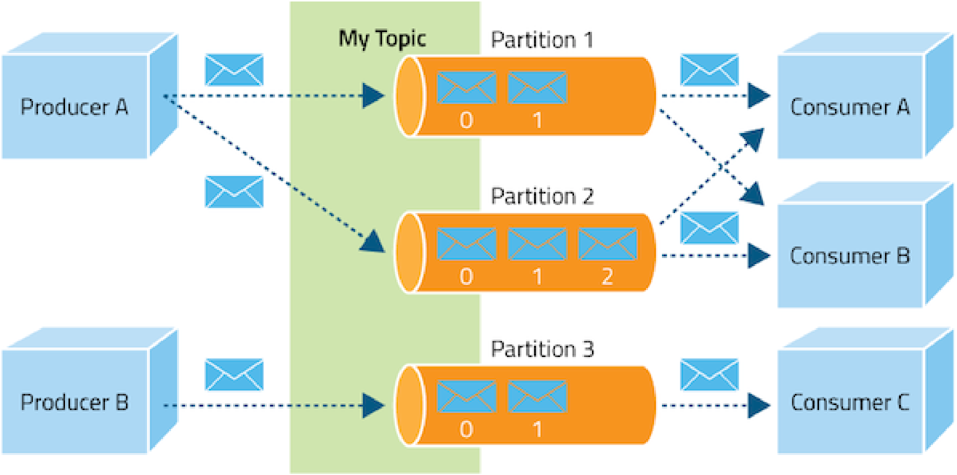
Table of Contents

Apache Kafka Overview

Part of the Hadoop ecosystem, Apache Kafka is a distributed commit log service that functions much like a publish/subscribe messaging system, but with better throughput, built-in partitioning, replication, and fault tolerance. Increasingly popular for log collection and stream processing, it is often (but not exclusively) used in tandem with Apache Hadoop, Apache Storm, and Spark Streaming.

A log can be considered as a simple storage abstraction. Because newer entries are appended to the log over time, from left to right, the log entry number is a convenient proxy for a timestamp. Conceptually, a log can be thought of as a time-sorted file or table.

Kafka integrates this unique abstraction with traditional publish/subscribe messaging concepts (such as producers, consumers, and brokers), parallelism, and enterprise features for improved performance and fault tolerance. The result is an architecture that, at a high level, looks like the following figure. (A topic is a category of messages that share similar characteristics.)



Kafka provides the following:

- Persistent messaging with O(1) disk structures that provide constant time performance, even with terabytes of stored messages.
- High throughput, supporting hundreds of thousands of messages per second, even with modest hardware.
- Explicit support for partitioning messages over Kafka servers and distributing consumption over a cluster of consumer machines while maintaining per-partition ordering semantics.
- Support for parallel data load into Hadoop.

Cloudera Distribution of Kafka Release Notes

New Features in Cloudera Distribution of Kafka 1.2.0

This release fixes some important issues; for details, see Issues Fixed in Cloudera Distribution of Kafka 1.2.0 on page 6.

New Features in Cloudera Distribution of Kafka 1.1.0

- **New producer**

  The new Kafka producer added in Cloudera Distribution of Kafka 1.1.0 combines features of the existing synchronous and asynchronous producers. Send requests are batched, allowing the new producer to perform as well as the asynchronous producer under load. Every send request returns a response object that can be used to retrieve status and exceptions.

- **Ability to delete topics**

  You can now delete topics using the `kafka-topics --delete` command.

- **Offset management**

  In previous versions, consumers that wanted to keep track of which messages were consumed did so by updating the offset of the last consumed message in Zookeeper. With this new feature, Kafka itself tracks the offsets. Using offset management can significantly improve consumer performance.

- **Automatic leader rebalancing**

  Each partition starts with a randomly selected leader replica that handles requests for that partition. When a cluster first starts, the leaders are evenly balanced among nodes. When a broker restarts, leaders from that broker are distributed to other brokers, which results in an unbalanced distribution. With this feature enabled, leaders are assigned to the original replica after a restart.

- **Connection quotas**

  Kafka administrators can limit the number of connections allowed from a single IP address. By default, this limit is 10 connections per IP address. This prevents misconfigured or malicious clients from destabilizing a Kafka broker by opening a large number of connections and using all available file handles.

Known Issues in Cloudera Distribution of Kafka 1.2.0

— High CPU utilization

Brokers with high partition count (approximately 2000) experience high CPU usage.

Bug: KAFKA-1952

Severity: Medium

Workaround: None

— NPE in Flume Kafka Source

> **Note:** Although not an issue with Apache Kafka, the following issue applies to Kafka sinks in Flume.

A Kafka source in Flume throws a NullPointerException if it processes a message with no key.

Bug: FLUME-2578

Severity: Medium

Workaround: Ensure that all messages have a key.

Issues Fixed in Cloudera Distribution of Kafka 1.2.0

Upstream Issues Fixed

The following upstream issues are fixed in Apache Kafka 1.2.0:

- KAFKA-1642 - [Java New Producer Kafka Trunk] CPU Usage Spike to 100% when network connection is lost
- KAFKA-1650 - avoid data loss when mirror maker shutdown uncleanly
- KAFKA-1797 - add the serializer/deserializer api to the new java client -
- KAFKA-1667 - topic-level configuration not validated
- KAFKA-1815 - ServerShutdownTest fails in trunk
- KAFKA-1861 - Publishing kafka-client:test in order to utilize the helper utils in TestUtils
- KAFKA-1729 - Add constructor to javaapi to allow constructing explicitly versioned offset commit requests
- KAFKA-1902 - fix MetricName so that Yammer reporter can work correctly
- KAFKA-1890 - Fix bug preventing Mirror Maker from successful rebalance
- KAFKA-1891 - MirrorMaker hides consumer exception - making troubleshooting challenging
- KAFKA-1706 - Add a byte bounded blocking queue utility
- KAFKA-1879 - Log warning when receiving produce requests with acks > 1
- KAFKA-1876 - pom file for scala 2.11 should reference a specific version
- KAFKA-1761 - num.partitions documented default is 1 while actual default is 2
- KAFKA-1210 - Windows Bat files are not working properly
- KAFKA-1864 - Revisit defaults for the internal offsets topic
- KAFKA-1870 - Cannot commit with simpleConsumer on Zookeeper only with Java API
- KAFKA-1868 - ConsoleConsumer shouldn't override dual.commit.enabled to false if not explicitly set
- KAFKA-1841 - OffsetCommitRequest API - timestamp field is not versioned
- KAFKA-1723 - make the metrics name in new producer more standard
- KAFKA-1819 Cleaner gets confused about deleted and re-created topics
- KAFKA-1851 - OffsetFetchRequest returns extra partitions when input only contains unknown partitions
- KAFKA-1512 - Fixes for limit the maximum number of connections per ip address
- KAFKA-1624 - bump up default scala version to 2.11.4 to compile with java 8
- KAFKA-742 - Existing directories under the Kafka data directory without any data cause process to not start
- KAFKA-1698 - Validator.ensureValid() only validates default config value
- KAFKA-1799 - ProducerConfig.METRIC_REPORTER_CLASSES_CONFIG doesn't work
- KAFKA-1743 - ConsumerConnector.commitOffsets in 0.8.2 is not backward compatible
- KAFKA-1769 - javadoc should only include client facing packages
- KAFKA-1481 - Stop using dashes AND underscores as separators in MBean names
- KAFKA-1721 - Snappy compressor is not thread safe
- KAFKA-1764 - ZookeeperConsumerConnector should not put multiple shutdown commands to the same data chunk queue
- KAFKA-1733 - Producer.send will block indeterminately when broker is unavailable
- KAFKA-1742 - ControllerContext removeTopic does not correctly update state
- KAFKA-1738 - Partitions for topic not created after restart from forced shutdown
- KAFKA-1647 - Replication offset checkpoints (high water marks) can be lost on hard kills and restarts
- KAFKA-1732 - DumpLogSegments tool fails when path has a '.'

Cloudera Distribution of Kafka Version and Packaging Information

Examples of Versions

Cloudera packages are designed to be transparent and easy to understand. Cloudera Distribution of Kafka package versions are labeled using the following format:

```
base_version+cloudera_version+patch_level
```

where,

- `base_version` is the version of the open-source component included in the Cloudera package
- `cloudera_version` is the version of the Cloudera package
- `patch_level` is the number of source commits applied on top of the base version forked from the Apache Kafka branch. Note that the number of commits does not indicate the number of functional changes or bug fixes in the release. For example, a commit may be used to amend a version number or make other non-functional changes.

Cloudera Distribution of Kafka Versions

Table 1: Cloudera Distribution of Kafka Version Information

Cloudera Distribution of Kafka Version	Component	Version	Release Notes	Custom Service Descriptor	Parcel Repository
1.2.0	Apache Kafka	0.8.2.0+kafka1.2.0+57	Release notes	Cloudera Distribution of Kafka 1.2.0 CSD	Cloudera Distribution of Kafka 1.2.0 Parcel Repository

Installing Kafka

> **Important:** As of February 1, 2021, all downloads of CDK, CDH, and Cloudera Manager require a username and password and use a modified URL. You must use the modified URL, including the username and password when downloading the repository contents described below. You may need to upgrade Cloudera Manager to a newer version that uses the modified URLs.
>
> This can affect new installations, upgrades, adding new hosts to a cluster, downloading a new parcel, and adding a new cluster.
>
> For more information, see Updating an existing CDH/Cloudera Manager deployment to access downloads with authentication.

> **Warning:** This version of Apache Kafka is only supported on Cloudera Manager 5.2.0 and higher on a parcel-deployed cluster. Do not use it with lower versions of Cloudera Manager or CDH or on a cluster deployed using packages or a tarball.

Kafka is distributed in a parcel that is independent of the CDH parcel and integrates with Cloudera Manager using a Custom Service Descriptor (CSD).

> **Note:** If you have installed a Cloudera Labs version of Kafka, you must download a new CSD and parcel. The Cloudera Labs CSD cannot install the GA Kafka parcel.

To install Apache Kafka:

1. Download the Kafka CSD here.
2. Install the CSD into Cloudera Manager as instructed in Custom Service Descriptor Files. This adds a new parcel repository to your Cloudera Manager configuration. The CSD can only be installed on parcel-deployed clusters.
3. Download, distribute, and activate the Kafka parcel, following the instructions in Managing Parcels. After you activate the Kafka parcel, Cloudera Manager prompts you to restart the cluster. Click the **Close** button to ignore this prompt. You *do not* need to restart the cluster after installing Kafka.
4. Add the Kafka service to your cluster, following the instructions in Adding a Service.

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

Kafka Command-line Tools

Important Kafka command-line tools are located in `/usr/bin`:

- `kafka-topics`

  Create, alter, list, and describe topics. For example:

```
$ /usr/bin/kafka-topics --list --zookeeper zk01.example.com:2181
sink1
t1
t2
```

- `kafka-console-consumer`

  Read data from a Kafka topic and write it to standard output. For example:

```
$ /usr/bin/kafka-console-consumer --zookeeper zk01.example.com:2181 --topic t1
```

- `kafka-console-producer`

  Read data from standard output and write it to a Kafka topic. For example:

```
$ /usr/bin/kafka-console-producer --broker-list
kafka02.example.com:9092,kafka03.example.com:9092 --topic t1
```

- `kafka-consumer-offset-checker`

  Check the number of messages read and written, as well as the lag for each consumer in a specific consumer group. For example:

```
$ /usr/bin/kafka-consumer-offset-checker --group flume --topic t1 --zookeeper
zk01.example.com:2181
```

Logs

The Kafka parcel is configured to log all Kafka log messages to a single file, `/var/log/kafka/server.log` by default. You can view, filter, and search this log using Cloudera Manager.

For debugging purposes, you can create a separate file with TRACE level logs of a specific component (such as the controller) or the state changes.

To do so, use the **Kafka broker Logging Advanced Configuration Snippet (Safety Valve)** field in Cloudera Manager (**Kafka Service** > **Configuration** > **Kafka broker Default Group** > **Advanced**) to add new appenders to `log4j`. For example, to restore the default Apache Kafka `log4j` configuration, copy the following into the safety valve:

```
log4j.appender.kafkaAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.kafkaAppender.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.kafkaAppender.File=${log.dir}/kafka_server.log
log4j.appender.kafkaAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.kafkaAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.appender.stateChangeAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.stateChangeAppender.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.stateChangeAppender.File=${log.dir}/state-change.log
log4j.appender.stateChangeAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.stateChangeAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.appender.requestAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.requestAppender.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.requestAppender.File=${log.dir}/kafka-request.log
log4j.appender.requestAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.requestAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.appender.cleanerAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.cleanerAppender.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.cleanerAppender.File=${log.dir}/log-cleaner.log
log4j.appender.cleanerAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.cleanerAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.appender.controllerAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.controllerAppender.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.controllerAppender.File=${log.dir}/controller.log
log4j.appender.controllerAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.controllerAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

# Turn on all our debugging info
#log4j.logger.kafka.producer.async.DefaultEventHandler=DEBUG, kafkaAppender
#log4j.logger.kafka.client.ClientUtils=DEBUG, kafkaAppender
#log4j.logger.kafka.perf=DEBUG, kafkaAppender
#log4j.logger.kafka.perf.ProducerPerformance$ProducerThread=DEBUG, kafkaAppender
#log4j.logger.org.I0Itec.zkclient.ZkClient=DEBUG
log4j.logger.kafka=INFO, kafkaAppender

log4j.logger.kafka.network.RequestChannel$=WARN, requestAppender
```

```
log4j.additivity.kafka.network.RequestChannel$=false

#log4j.logger.kafka.network.Processor=TRACE, requestAppender
#log4j.logger.kafka.server.KafkaApis=TRACE, requestAppender
#log4j.additivity.kafka.server.KafkaApis=false
log4j.logger.kafka.request.logger=WARN, requestAppender
log4j.additivity.kafka.request.logger=false

log4j.logger.kafka.controller=TRACE, controllerAppender
log4j.additivity.kafka.controller=false

log4j.logger.kafka.log.LogCleaner=INFO, cleanerAppender
log4j.additivity.kafka.log.LogCleaner=false

log4j.logger.state.change.logger=TRACE, stateChangeAppender
log4j.additivity.state.change.logger=false
```

Alternatively, you can add only the appenders you need.

More Information

For more information, see the official Kafka documentation.

When using Kafka, consider the following:

- Use Cloudera Manager to start and stop Kafka and ZooKeeper services. Do not use the `kafka-server-start`, `kafka-server-stop`, `zookeeper-server-start`, and `zookeeper-server-stop` commands.
- All Kafka command-line tools are located in `/opt/cloudera/parcels/KAFKA/lib/kafka/bin/`.
- Set the `JAVA_HOME` environment variable to your JDK installation directory before using the command-line tools. For example:

```
export JAVA_HOME=/usr/java/jdk1.7.0_55-cloudera
```

Kafka Administration

This section describes how to configure and manage Kafka, including performance tuning and high availability considerations.

Using Kafka with Flume

In CDH 5.2.0 and higher, Flume contains a Kafka source and sink. Use these to stream data from Kafka to Hadoop or from any Flume source to Kafka.

> **Important:** 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 source and a sink, use an interceptor to modify the event header and set a different topic.

Kafka Source

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**.

Table 2: Kafka Source Properties

Property Name	Default Value	Description
type		Must be set to `org.apache.flume.source.kafka.KafkaSource`.

Property Name	Default Value	Description
zookeeperConnect		The URI of the ZooKeeper server or quorum used by Kafka. This can be a single node (for example, `zk01.example.com:2181`) or a comma-separated list of nodes 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 Kafka documentation for the full list of Kafka consumer properties.

Tuning Notes

The Kafka source overrides two Kafka consumer parameters:

1. `auto.commit.enable` is set to `false` by the source, and every batch is committed. For improved performance, set this to `true` using the `kafka.auto.commit.enable` setting. This 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 to a higher value can reduce CPU utilization due to less frequent polling, but introduces latency in writing batches to the channel.

Kafka Sink

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**.

Table 3: Kafka Sink Properties

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 Cloudera recommends that 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.
requiredAcks	1	The number of replicas that must acknowledge a message before it is written successfully. Possible values are `0` (do not wait for an acknowledgement), `1` (wait for the leader to acknowledge only), and `-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 Kafka documentation 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 Channel

CDH 5.3 and higher 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.transactionCapacity = 1000
tier1.channels.channel1.brokerList = kafka02.example.com:9092,kafka03.example.com:9092
```

```
tier1.channels.channel1.topic = channel2
tier1.channels.channel1.zookeeperConnect = zk01.example.com:2181
tier1.channels.channel1.parseAsFlumeEvent = true

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**.

Table 4: Kafka Channel Properties

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 Cloudera recommends that you specify at least two for high availability.
zookeeperConnect		The URI of the ZooKeeper server or quorum used by Kafka. This can be a single node (for example, `zk01.example.com:2181`) or a comma-separated list of nodes 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.
readSmallestOffset	false	If `true`, reads all data in the topic. If `false`, reads only data written after the channel has started. Only used when `parseAsFlumeEvent` is `false`.
kafka.consumer.timeout.ms	100	Polling interval when writing to the sink.
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 Kafka documentation for the full list of Kafka producer properties.

Using Kafka with Spark Streaming

For information on how to configure Spark Streaming to receive data from Kafka, see the Spark Streaming + Kafka Integration Guide.

Validating Kafka Integration with Spark Streaming

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

```
/opt/cloudera/parcels/CDH/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.
- *<topic>* - 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.

Building Your Own Spark Streaming Application

To deploy your own application, follow these steps:

1. Build an uber-jar (a single JAR that includes the application and all dependencies, such as Kafka and ZooKeeper) using a Maven plugin such as Assembly or Shade.

   Download an example application here for reference. Kafka and ZooKeeper are specified as dependencies, even though they are not used directly in the code.

2. Build the project using `mvn install` and copy the uber-jar to the cluster.
3. To run the application, use `spark-submit`:

```
spark-submit --master <master> --class <application_main_class> <JAR> <parameters>
```

See the Spark documentation for information on which master to use and how to specify it.

To run the provided example application on a local master, run the following:

```
spark-submit --master local[*] --class com.shapira.examples.streamingavg.StreamingAvg
uber-StreamingAvg-1.0-SNAPSHOT.jar localhost:2181/kafka group1 topic3 1
```

Kafka High Availability and Consistency

To achieve high availability and consistency targets, adjust the following parameters to meet your requirements:

Replication Factor

The default replication factor for new topics is 1. For highly-available production systems, Cloudera recommends setting the replication factor to at least 3. This requires at least 3 Kafka brokers.

To change the replication factor, navigate to **Kafka Service** > **Configuration** > **Service-Wide**. Set **Replication factor** to 3, click **Save Changes**, and restart the Kafka service.

Unclean Leader Election

With unclean leader election disabled, if a broker containing the leader replica for a partition becomes unavailable, and no in-sync replica exists to replace it, the partition becomes unavailable until the leader replica or another in-sync replica is back online. Enable unclean leader election to allow an out-of-sync replica to become the leader and preserve the availability of the partition. With unclean leader election, messages that were not synced to the new leader are lost. This provides balance between consistency (guaranteed message delivery) and availability.

To enable unclean leader election, navigate to **Kafka Service** > **Configuration** > **Service-Wide**. Check the box labeled **Enable unclean leader election**, click **Save Changes**, and restart the Kafka service.

Acknowledgements

When writing or configuring a Kafka producer, you can choose how many replicas commit a new message before the message is acknowledged using the `requiredAcks` property (see Table 3: Kafka Sink Properties on page 13 for details).

Set `requiredAcks` to `0` (immediately acknowledge the message without waiting for any brokers to commit), `1` (acknowledge after the leader commits the message), or `-1` (acknowledge after all in-sync replicas are committed) according to your requirements. Setting `requiredAcks` to `-1` provides the highest consistency guarantee at the expense of slower writes to the cluster.

Minimum In-sync Replicas

You can also set the minimum number of in-sync replicas that must be available for the producer to successfully send messages to a partition using the `min.insync.replicas` setting. If `min.insync.replicas` is set to 2 and `requiredAcks` is set to `-1`, each message must be written successfully to at least two replicas. This guarantees that the message is not lost unless both hosts crash.

It also means that if one of the nodes crashes, the partition is no longer available for writes. Similarly to the unclean leader election configuration, setting `min.insync.replicas` is a balance between higher consistency (requiring writes to more than one broker) and higher availability (allowing writes when fewer brokers are available).

To configure `min.insync.replicas` at the cluster level, navigate to **Kafka Service** > **Configuration** > **Service-Wide**. Set **Minimum number of replicas in ISR** to the desired value, click **Save Changes**, and restart the Kafka service.

To set this parameter on a per-topic basis, navigate to **Kafka Service** > **Configuration** > **Kafka broker Default Group** > **Advanced**, and add the following to the **Kafka broker Advanced Configuration Snippet (Safety Valve) for kafka.properties**:

```
min.insync.replicas.per.topic=topic_name_1:value,topic_name_2:value
```

Replace *topic_name_n* with the topic names, and replace *value* with the desired minimum number of in-sync replicas.

You can also set this parameter using the `/usr/bin/kafka-topics --alter` command for each topic. For example:

```
/usr/bin/kafka-topics --alter --zookeeper zk01.example.com:2181 --topic topicname \
--config min.insync.replicas=2
```

Kafka MirrorMaker

Kafka mirroring enables maintaining a replica of an existing Kafka cluster. For production use, specify the `--no.data.loss` parameter. This automatically sets producer parameters to avoid losing data in unexpected events. Data duplication is possible in some scenarios. For example, if MirrorMaker crashes, it duplicates messages since its previous checkpoint.

Checkpoint frequency is controlled with the `offset.commit.interval.ms` argument. This balances performance and number of duplicates. Committing more frequently is slower, but results in fewer duplicates.

Kafka Performance and Resource Considerations

Kafka is optimized for small messages. According to benchmarks, the best performance occurs with 1 KB messages. Larger messages (for example, 10 MB to 100 MB) can decrease throughput and significantly impact operations.

Partitions and Memory Usage

Brokers allocate a buffer the size of `replica.fetch.max.bytes` for each partition they replicate. If `replica.fetch.max.bytes` is set to 1 MiB and you have 1000 partitions, about 1 GiB of RAM is required. Ensure that the number of partitions multiplied by the size of the largest message does not exceed available memory.

The same consideration applies for the consumer `fetch.message.max.bytes` setting. Ensure that you have enough memory for the largest message for each partition the consumer replicates. When using larger messages, you may need to use fewer partitions or provide more RAM.

Garbage Collection

Large messages can cause longer garbage collection (GC) pauses as brokers allocate large chunks. Monitor the GC log and the server log. If long GC pauses cause Kafka to abandon the ZooKeeper session, you may need to configure longer timeout values for `zookeeper.session.timeout.ms`.

Handling Large Messages

If you need to accommodate 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.codec` and `compressed.topics` producer configuration parameters to enable compression. Both Gzip 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 following configuration parameters to match your requirements:

Broker Configuration

- `message.max.bytes`

  Maximum message size the broker will accept. Must be smaller than the consumer `fetch.message.max.bytes`, or the consumer cannot consume the message.

  Default value: `1000000` (1 MB)

- `log.segments.bytes`

  Size of a Kafka data file. Must be larger than any single message.

  Default value: `1073741824` (1 GiB)

- `replica.fetch.max.bytes`

  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.

  Default value: `1048576` (1 MiB)

Consumer Configuration

- `fetch.message.max.bytes`

  Maximum message size a consumer can read. Must be at least as large as `message.max.bytes`.

Default value: `1048576` (1 MiB)

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.
```