

cloudera[®]

Cloudera Distribution of Apache Kafka

Important Notice

© 2010-2021 Cloudera, Inc. All rights reserved.

Cloudera, the Cloudera logo, and any other product or service names or slogans contained in this document are trademarks of Cloudera and its suppliers or licensors, and may not be copied, imitated or used, in whole or in part, without the prior written permission of Cloudera or the applicable trademark holder. If this documentation includes code, including but not limited to, code examples, Cloudera makes this available to you under the terms of the Apache License, Version 2.0, including any required notices. A copy of the Apache License Version 2.0, including any notices, is included herein. A copy of the Apache License Version 2.0 can also be found here: <https://opensource.org/licenses/Apache-2.0>

Hadoop and the Hadoop elephant logo are trademarks of the Apache Software Foundation. All other trademarks, registered trademarks, product names and company names or logos mentioned in this document are the property of their respective owners. Reference to any products, services, processes or other information, by trade name, trademark, manufacturer, supplier or otherwise does not constitute or imply endorsement, sponsorship or recommendation thereof by us.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Cloudera.

Cloudera may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Cloudera, the furnishing of this document does not give you any license to these patents, trademarks copyrights, or other intellectual property. For information about patents covering Cloudera products, see <http://tiny.cloudera.com/patents>.

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

Cloudera Distribution of Apache Kafka.....	5
---	----------

Cloudera Distribution of Apache Kafka Release Notes.....	6
---	----------

New Features in Cloudera Distribution of Apache Kafka.....	6
Requirements.....	7
<i>Supported Operating Systems.....</i>	<i>7</i>
<i>Supported JDK Versions.....</i>	<i>8</i>
Known issues.....	8
Fixed Issues.....	8
<i>Issues Fixed in Cloudera Distribution of Apache Kafka 1.4.0.....</i>	<i>8</i>
<i>Issues Fixed in Cloudera Distribution of Apache Kafka 1.3.2.....</i>	<i>9</i>
<i>Issues Fixed in Cloudera Distribution of Apache Kafka 1.3.1.....</i>	<i>9</i>
<i>Issues Fixed in Cloudera Distribution of Apache Kafka 1.3.0.....</i>	<i>9</i>
<i>Issues Fixed in Cloudera Distribution of Apache Kafka 1.2.0.....</i>	<i>9</i>

Cloudera Distribution of Apache Kafka Version and Packaging Information.....	11
---	-----------

Examples of Versions.....	11
Cloudera Distribution of Apache Kafka Versions.....	11
Maven Artifacts for Kafka.....	12

Installing and Upgrading Kafka.....	13
--	-----------

Installing Kafka.....	13
<i>Installing the Kafka CSD (Cloudera Manager 5.2 and 5.3).....</i>	<i>13</i>
<i>Installing Kafka from a Parcel.....</i>	<i>13</i>
<i>Installing Kafka from a Package.....</i>	<i>14</i>
Upgrading Kafka or Cloudera Manager.....	15
<i>Upgrading Kafka Using Parcels.....</i>	<i>15</i>
<i>Upgrading Cloudera Manager Without Upgrading Kafka.....</i>	<i>15</i>
<i>Upgrading Both Cloudera Manager and Kafka.....</i>	<i>15</i>
<i>Upgrading Cloudera Manager Before Installing Kafka.....</i>	<i>16</i>
Migrating from Apache Kafka to Cloudera Distribution of Kafka.....	16
<i>Migration Steps.....</i>	<i>16</i>

Using Kafka.....	18
-------------------------	-----------

Using Kafka Command-line Tools.....18
Using Kafka with Spark Streaming.....18
Using Kafka with Flume.....19
Additional Information.....23

Kafka Administration.....24

High Availability and Consistency.....24
Performance and Resource Considerations.....25
Partitions and Memory Usage.....25
Garbage Collection.....26
Handling Large Messages.....26
Tuning Kafka for Optimal Performance.....27
Metrics.....29
Kafka Cluster Metrics.....29
Kafka Broker Metrics in Cloudera Manager.....30
Logs.....31

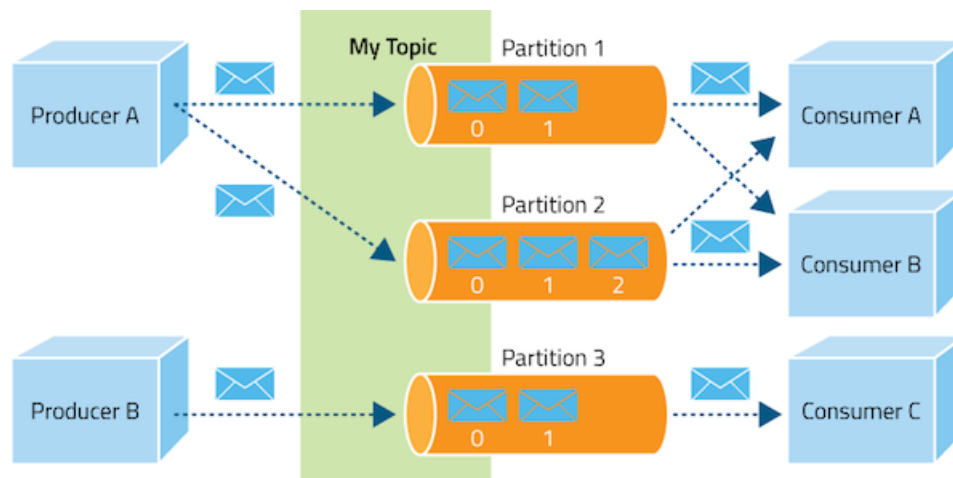
Appendix: Apache License, Version 2.0.....33

Cloudera Distribution of Apache Kafka

Part of the Hadoop ecosystem, Cloudera Distribution of 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 Apache Kafka Release Notes

New Features in Cloudera Distribution of Apache Kafka

New Features in Cloudera Distribution of Apache Kafka 1.4.0

- Kafka 1.4.0 is distributed as a package as well as a parcel. See [Cloudera Distribution of Apache Kafka Version and Packaging Information](#) on page 11.

New Features in Cloudera Distribution of Apache Kafka 1.3.2

- **RHEL 7.1**
Kafka 1.3.2 supports RHEL 7.1. See [Supported Operating Systems](#) on page 7

New features in Cloudera Distribution of Apache Kafka 1.3.0

- **Metrics Reporter**
Cloudera Manager now displays Kafka metrics. Use the values to identify current performance issues and plan enhancements to handle anticipated changes in workload. See [Metrics](#) on page 29.
- **MirrorMaker configuration**
Cloudera Manager allows you to configure the Kafka MirrorMaker cross-cluster replication service. You can add a MirrorMaker role and use it to replicate to a machine in another cluster. See [Kafka MirrorMaker](#).

New Features in Cloudera Distribution of Apache Kafka 1.1.0

- **New producer**
The new Kafka producer added in Cloudera Distribution of Apache 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 hosts. 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.

Requirements

The following sections describe the supported versions of operating systems and JDK for Cloudera Distribution of Apache Kafka 1.4.0.

Supported Operating Systems

Kafka 1.4.0 provides parcels for RHEL-compatible, SLES, Ubuntu, and Debian systems as described below.

Operating System	Version	Packages
Red Hat Enterprise Linux (RHEL)-compatible		
RHEL	5.7	64-bit
	5.10	64-bit
	6.4	64-bit
	6.5	64-bit
	6.5 in SE Linux mode	64-bit
	6.6	64-bit
	7.1	64-bit
CentOS	5.7	64-bit
	5.10	64-bit
	6.4	64-bit
	6.5	64-bit
	6.5 in SE Linux mode	64-bit
	6.6	64-bit
	6.6 in SE Linux mode	64-bit
	6.7	64-bit
	7.1	64-bit
Oracle Linux with default kernel and Unbreakable Enterprise Kernel	5.6 (UEK R2)	64-bit
	6.4 (UEK R2)	64-bit
	6.5 (UEK R2, UEK R3)	64-bit
	6.6 (UEK R3)	64-bit
	7.1	64-bit
SLES		
SUSE Linux Enterprise Server (SLES)	11 with Service Pack 2	64-bit
SUSE Linux Enterprise Server (SLES)	11 with Service Pack 3	64-bit
Ubuntu/Debian		
Ubuntu	Precise (12.04) - Long-Term Support (LTS)	64-bit
	Trusty (14.04) - Long-Term Support (LTS)	64-bit

Operating System	Version	Packages
Debian	Wheezy (7.0, 7.1)	64-bit



Note:

- If you are using an operating system that is not supported by Cloudera packages, you can also download source tarballs from [Downloads](#).
- The supported use of RHEL 7 is as follows:
 1. Only RHEL 7.1 is supported with Kafka 1.3.2. RHEL 7.0 is not supported.
 2. Only a new installation of RHEL 7.1 is supported. Upgrades from RHEL 6.X to RHEL 7.1 are not supported. For more information, see <https://access.redhat.com/solutions/21964>.

Supported JDK Versions



Important: There is one exception to the minimum supported and recommended JDK versions in the following table. If Oracle releases a security patch that affects server-side Java before the next minor release of Cloudera products, the Cloudera support policy covers customers using the patch.

Kafka is supported with the versions shown in the following table:

Minimum Supported Version	Recommended Version	Exceptions
1.7.0_25	1.7.0_80	None
1.8.0_31	1.8.0_60	Cloudera recommends that you do not use JDK 1.8.0_40.

Known issues

If you use Cloudera Distribution of Kafka 1.2 with Cloudera Manager 5.4, you must disable monitoring. See [Upgrading Cloudera Manager Without Upgrading Kafka](#) on page 15.

Fixed Issues

The following upstream issues are fixed in each release of Cloudera Distribution of Apache Kafka.

Issues Fixed in Cloudera Distribution of Apache Kafka 1.4.0

- [KAFKA-1664](#): Kafka does not properly parse multiple ZK nodes with non-root chroot.
- [KAFKA-1994](#): Evaluate performance effect of chroot check on Topic creation.
- [KAFKA-2002](#): It does not work when kafka_mx4jenable is false.
- [KAFKA-2024](#): Cleaner can generate unindexable log segments.
- [KAFKA-2048](#): java.lang.IllegalMonitorStateException thrown in AbstractFetcherThread when handling error returned from simpleConsumer.
- [KAFKA-2050](#): Avoid calling .size() on java.util.ConcurrentLinkedQueue.
- [KAFKA-2088](#): kafka-console-consumer.sh should not create zookeeper path when no brokers found and chroot was set in zookeeper.connect.
- [KAFKA-2118](#): Cleaner cannot clean after shutdown during replaceSegments.
- [KAFKA-2477](#): Fix a race condition between log append and fetch that causes OffsetOutOfRangeException.
- [KAFKA-2633](#): Default logging from tools to Stderr.

Issues Fixed in Cloudera Distribution of Apache Kafka 1.3.2

- [KAFKA-1057](#): Trim whitespaces from user specified configs
- [KAFKA-1641](#): Log cleaner exits if last cleaned offset is lower than earliest offset
- [KAFKA-1702](#): Messages silently lost by the (old) producer
- [KAFKA-1758](#): corrupt recovery file prevents startup
- [KAFKA-1836](#): metadata.fetch.timeout.ms set to zero blocks forever
- [KAFKA-1866](#): LogStartOffset gauge throws exceptions after log.delete()
- [KAFKA-1883](#): NullPointerException in RequestSendThread
- [KAFKA-1896](#): Record size function of record in mirror maker hit NPE when the message value is null.
- [KAFKA-2012](#): Broker should automatically handle corrupt index files
- [KAFKA-2096](#): Enable keepalive socket option for broker to prevent socket leak
- [KAFKA-2114](#): Unable to set default min.insync.replicas
- [KAFKA-2189](#): Snappy compression of message batches less efficient in 0.8.2.1
- [KAFKA-2234](#): Partition reassignment of a nonexistent topic prevents future reassignments
- [KAFKA-2235](#): LogCleaner offset map overflow
- [KAFKA-2336](#): Changing offsets.topic.num.partitions after the offset topic is created breaks consumer group partition assignment
- [KAFKA-2393](#): Correctly Handle InvalidTopicException in KafkaApis.getTopicMetadata()
- [KAFKA-2406](#): ISR propagation should be throttled to avoid overwhelming controller
- [KAFKA-2407](#): Only create a log directory when it will be used
- [KAFKA-2437](#): Fix ZookeeperLeaderElector to handle node deletion correctly
- [KAFKA-2468](#): SIGINT during Kafka server startup can leave server deadlocked
- [KAFKA-2504](#): Stop logging WARN when client disconnects

Issues Fixed in Cloudera Distribution of Apache Kafka 1.3.1

- [KAFKA-972](#) - MetadataRequest returns stale list of brokers
- [KAFKA-1367](#) - Broker topic metadata not kept in sync with ZooKeeper
- [KAFKA-1867](#) - liveBroker list not updated on a cluster with no topics
- [KAFKA-2308](#) - New producer + Snappy face un-compression errors after broker restart
- [KAFKA-2317](#) - De-register isrChangeNotificationListener on controller resignation
- [KAFKA-2337](#) - Verify that metric names will not collide when creating new topics

Issues Fixed in Cloudera Distribution of Apache Kafka 1.3.0

- [KAFKA-2009](#) - Fix UncheckedOffset.removeOffset synchronization and trace logging issue in mirror maker
- [KAFKA-1984](#) - Java producer may miss an available partition
- [KAFKA-1971](#) - Starting a broker with a conflicting id will delete the previous broker registration
- [KAFKA-1952](#) - High CPU Usage in 0.8.2 release
- [KAFKA-1919](#) - Metadata request issued with no backoff in new producer if there are no topics

Issues Fixed in Cloudera Distribution of 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 Apache Kafka Version and Packaging Information

Examples of Versions

Cloudera packages are designed to be transparent and easy to understand. Cloudera Distribution of Apache 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 can be used to amend a version number or make other non-functional changes.

Cloudera Distribution of Apache Kafka Versions

Table 1: Cloudera Distribution of Apache Kafka Version Information

Cloudera Distribution of Apache Kafka Version	Component	Version	Release Notes	Custom Service Descriptor	Parcel Repository
1.4.0	Apache Kafka	0.8.2.0+kafka1.4.0+127	Release notes	Included with Cloudera Manager 5.4.x and higher. See Installing Kafka	Cloudera Distribution of Apache Kafka 1.4.0 Parcel Repository
1.3.2	Apache Kafka	0.8.2.0+kafka1.3.2+116	Release notes	Included with Cloudera Manager 5.4.x and higher. See Installing Kafka	Cloudera Distribution of Apache Kafka 1.3.2 Parcel Repository
1.3.1	Apache Kafka	0.8.2.0+kafka1.3.1+80	Release notes	Included with Cloudera Manager 5.4.x and higher. See Installing Kafka	Cloudera Distribution of Apache Kafka 1.3.1 Parcel Repository
1.3.0	Apache Kafka	0.8.2.0+kafka1.3.0+72	Release notes	Included with Cloudera Manager 5.4.x and higher. See Installing Kafka	Cloudera Distribution of Apache Kafka 1.3.0 Parcel Repository
1.2.0	Apache Kafka	0.8.2.0+kafka1.2.0+57	Release notes	Cloudera Distribution of	Cloudera Distribution of Apache Kafka

Cloudera Distribution of Apache Kafka Version and Packaging Information

Cloudera Distribution of Apache Kafka Version	Component	Version	Release Notes	Custom Service Descriptor	Parcel Repository
				Apache Kafka 1.2.0 CSD	1.2.0 Parcel Repository

Maven Artifacts for Kafka

The following table lists the project name, groupId, artifactId, and version required to access each Kafka artifact from a Maven POM.

Project	groupId	artifactId	version
Kafka	org.apache.kafka	kafka-clients	0.8.2.0-kafka-1.3.2
Kafka	org.apache.kafka	kafka-examples	0.8.2.0-kafka-1.3.2
Kafka	org.apache.kafka	kafka_2.10	0.8.2.0-kafka-1.3.2

Installing and Upgrading Kafka

[Required Role:](#) 



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](#).

The steps required to install or upgrade Kafka vary based on the version of Cloudera Manager you are using. This section describes several possible installation and upgrade scenarios.

Installing Kafka

[Required Role:](#) 

Kafka is distributed as a parcel, separate from the CDH parcel. It is also distributed as a package.

Cloudera Manager 5.4 or higher includes the Kafka Custom Service Descriptor (CSD). To install, you download Kafka using CM, then distribute Kafka to the cluster, activate the new parcel, and add the service to the cluster. For a list of available parcels and packages see [Cloudera Distribution of Apache Kafka Version and Packaging Information](#) on page 11

To install Kafka on a 5.2 or 5.3 cluster, you first install the Kafka CSD, and then use the CSD to install the Kafka parcel. See [Installing the Kafka CSD \(Cloudera Manager 5.2 and 5.3\)](#).

Kafka is supported only on Cloudera Manager 5.2 and higher. Do not use it with lower versions of Cloudera Manager or CDH.

If you installed a Cloudera Labs version of Kafka, you must download a new version. The Cloudera Labs CSD cannot install the GA Kafka parcel.

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

Installing the Kafka CSD (Cloudera Manager 5.2 and 5.3)

Cloudera Manager 5.4 and higher includes the Kafka CSD. Use the built-in CSD. Do not download a different version when using Cloudera Manager 5.4.

If you are using Cloudera Manager 5.2 or 5.3, download the CSD, and then add a new parcel repository to your Cloudera Manager configuration:

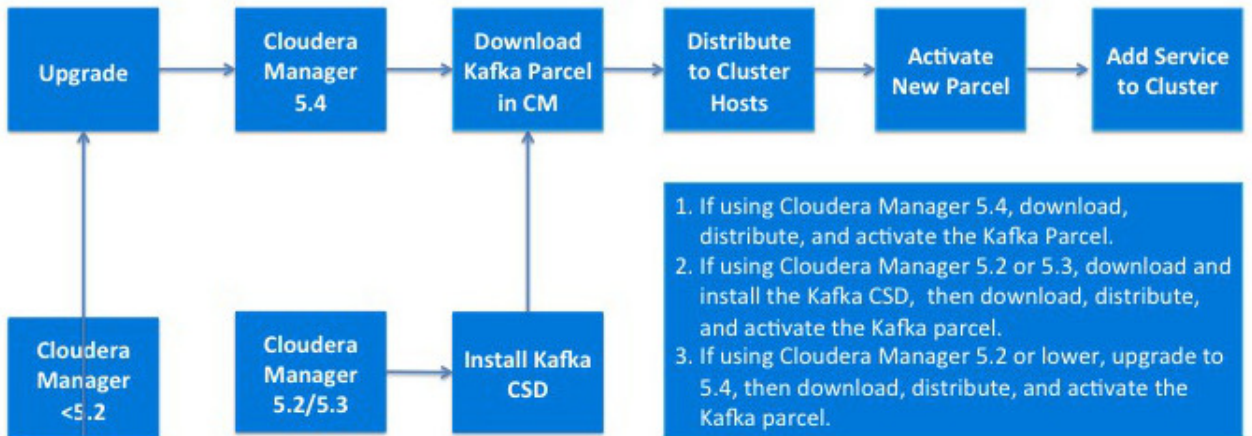
1. Download the Kafka CSD [here](#).
2. Install the Kafka CSD into Cloudera Manager. See [Custom Service Descriptor Files](#).

Installing Kafka from a Parcel

1. In Cloudera Manager, download the Kafka parcel, distribute the parcel to the hosts in your cluster, and then activate the parcel. See [Managing Parcels](#). After you activate the Kafka parcel, Cloudera Manager prompts you to restart the cluster. You *do not* need to restart the cluster after installing Kafka. Click **Close** to ignore this prompt.

Installing and Upgrading Kafka

2. Add the Kafka service to your cluster. See [Adding a Service](#).



Installing Kafka from a Package

You install the Kafka package from the command line.

1. Navigate to the `/etc/repos.d` directory.
2. Use `wget` to download the Kafka repository. See [Cloudera Distribution of Apache Kafka Version and Packaging Information](#) on page 11.
3. Install Kafka using the appropriate commands for your operating system.

Operating System	Commands
RHEL-compatible	<pre>\$ sudo yum clean all \$ sudo yum install kafka \$ sudo yum install kafka-server</pre>
SLES	<pre>\$ sudo zypper clean --all \$ sudo zypper install kafka \$ sudo zypper install kafka-server</pre>
Ubuntu or Debian	<pre>\$ sudo apt-get update \$ sudo apt-get install kafka \$ sudo apt-get install kafka-server</pre>

4. Edit `/etc/kafka/conf/server.properties` to ensure that the `broker.id` is unique for each node and broker in Kafka cluster, and `zookeeper.connect` points to same zookeeper for all nodes and brokers.
5. Start the Kafka server with the following command:

```
$ sudo service kafka-server start.
```

To verify all nodes are registered to same Zookeeper correctly, connect to Zookeeper using `zookeeper-client`.

```
$ zookeeper-client
$ ls /brokers/ids
```

You should be able to see all of the IDs for the brokers you have registered in your Kafka cluster.

To discover which node a particular ID is assigned, use the following command.

```
$ get /brokers/ids/<ID>
```

This command returns the host name of node assigned the ID you specify.

Upgrading Kafka or Cloudera Manager

Upgrading Kafka Using Parcels

[Required Role:](#) 



1. Download the Kafka parcel to Cloudera Manager, distribute the parcel to the hosts in your cluster, and then activate the new parcel. See [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.
2. Restart the Kafka service (either broker-by-broker or all at once). See [Starting, Stopping, and Restarting Services](#).

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



Note: These instructions assume that you are upgrading Kafka installed from a parcel. If you want to upgrade using a Kafka package, you first must uninstall parcel-based Kafka. See [Uninstalling an Add-on Service](#).

Upgrading Cloudera Manager Without Upgrading Kafka



1. Delete the previous Kafka CSD JAR from `/opt/cloudera/csd/`. See [Uninstalling an Add-on Service](#).
2. Upgrade [Cloudera Manager](#).
3. Disable Monitoring:
 - a. In Cloudera Manager, select your Kafka instance.
 - b. Click **Configuration**.
 - c. De-select the **Enable monitoring** checkbox.

Upgrading Both Cloudera Manager and Kafka



1. Delete the previous Kafka CSD JAR from `/opt/cloudera/csd/`. See [Uninstalling an Add-on Service](#).
2. Upgrade [Cloudera Manager](#).

Installing and Upgrading Kafka

3. Download the Kafka parcel to Cloudera Manager, distribute the parcel to the hosts in your cluster, and then activate the parcel. See [Managing Parcels](#). After you activate the Kafka parcel, Cloudera Manager prompts you to restart the cluster. You *do not* need to restart the cluster after installing Kafka. Click **Close** to ignore this prompt.
4. Restart the Kafka service (either broker-by-broker or all at once). See [Starting, Stopping, and Restarting Services](#).



Note: These instructions assume that you are upgrading Kafka installed from a parcel. If you want to upgrade using a Kafka package, you first must uninstall parcel-based Kafka. See [Uninstalling an Add-on Service](#).

Upgrading Cloudera Manager Before Installing Kafka

If you have never installed Kafka, you can upgrade Cloudera Manager and then install Kafka. See [Installing Kafka](#) on page 13.

Migrating from Apache Kafka to Cloudera Distribution of Kafka

This topic describes the required steps to migrate an existing Apache Kafka instance to Cloudera Distribution of Apache Kafka.

Assumptions

- This guide assumes you are migrating to a Kafka cluster managed by Cloudera Manager.
- This guide assumes you can plan a maintenance window for your migration.
- This guide assumes you are migrating from a compatible release version, as shown in the table below:

From Apache Kafka	To Cloudera Distribution of Kafka
0.8.x	1.x



Note: Note: Migration from Apache Kafka 0.7.x is not supported. If running Apache Kafka 0.7.x or earlier, you must first migrate to Apache Kafka 0.8.x.

Migration Steps

Cloudera suggests you migrate your system using the following procedure. The order in which you migrate components is significant. It is important to migrate brokers first, then migrate clients.

Before You Begin

1. Shut down all existing producers, consumers, MirrorMaker instances, and Kafka brokers.
2. If not already installed, install Cloudera Manager. See [Installing Cloudera Manager, CDH, and Managed Services](#).
 - a. Add the CDH and Kafka Parcels at installation time.
 - b. Do not add any services yet: those instructions are below. Skip the install page by clicking the Cloudera Manager **Home** button.

Step 1. Migrating Zookeeper

Kafka stores its metadata in ZooKeeper. As part of migrating to Cloudera Distribution of Kafka, you must also migrate your ZooKeeper instance to the supported version, included with CDH.

1. Shut down your existing ZooKeeper cluster.
2. Back up your `dataDir` and `dataLogDir` by copying them to another location or machine.
3. Add the ZooKeeper service to the cluster where you want to run Cloudera Kafka. See [Adding a Service](#).

4. Add the ZooKeeper role to all machines that were running ZooKeeper.
5. Set any custom configuration from your old `zoo.cfg` file in Cloudera Manager.
6. Make sure `dataDir` and `dataLogDir` match your old configuration. This is important because this is where all your data is stored.
7. Make sure the `zookeeper` user owns the files in the `dataDir` and `dataLogDir`. For example:

```
ex: chown -R zookeeper /var/lib/zookeeper
```

8. Start the new ZooKeeper service.
9. Use the `zookeeper-client` CLI to validate that some data exists. You should see nodes such as *brokers*, *consumers*, and *configs*. You might need to adjust your `chroot`. For example:

```
zookeeper-client -server hostname:port
                  ls /
```

Step 2. Migrating Kafka Brokers

1. All producers, consumers, and Kafka brokers should still be shut down.
2. Back up your `log.dirs` from the old broker machines by copying them to another location or machine.
3. Add the Kafka service to the cluster where you migrated ZooKeeper. See [Adding a Service](#).
4. Add the `broker` role to all machines that were running brokers.
5. Make sure the `kafka` user owns the `log.dirs` files. For example:

```
chown -R kafka /var/local/Kafka/data
```

6. Set any custom configuration from your old `server.properties` file in Cloudera Manager.
 - Make sure to override the `broker.id` on each node to match the configured value in your old configurations. This is important: if these values do not match, Kafka treats your brokers as new brokers and not your existing ones.
 - Make sure `log.dirs` and `zookeeper.chroot` matches your old configuration. This is important, because this is where all of your data and state information is stored.
7. Start the Kafka brokers using Cloudera Manager.

Step 3. Migrating MirrorMaker



Warning: Migrate downstream clusters first to avoid compatibility issues.

1. Add the MirrorMaker role to all machines that were running MirrorMaker before.
2. Set any custom configuration from your old `producer.properties` and `consumer.properties` files in Cloudera Manager.
3. Start the MirrorMaker instances using Cloudera Manager.

Step 4. Migrating Kafka Clients

Although Kafka might function with your existing clients, you must also upgrade all of your producers and consumers in order to have all Cloudera patches and bug fixes, and to have a fully supported system.

Migration requires that you change your Kafka dependencies from the Apache versions to the Cloudera versions, recompile your classes, and redeploy them. Use the Maven repository locations as described in [Maven Artifacts for Kafka](#) on page 12.

Using Kafka

This section describes ways you can use Kafka tools for data capture for analysis.

Using Kafka Command-line Tools

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
```

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
```

Using Kafka with Flume

In CDH 5.2 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.channell.capacity = 10000
tier1.channels.channell.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 = channell

```

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 <code>org.apache.flume.source.kafka.KafkaSource</code> .
zookeeperConnect		The URI of the ZooKeeper server or quorum used by Kafka. This can be a single host (for example, <code>zk01.example.com:2181</code>) or a comma-separated list of hosts in a ZooKeeper quorum (for example, <code>zk01.example.com:2181, zk02.example.com:2181, zk03.example.com:2181</code>).
topic		The Kafka topic from which this source reads messages. Flume supports only one topic per source.
<code>groupID</code>	<code>flume</code>	The unique identifier of the Kafka consumer group. Set the same <code>groupID</code> in all sources to indicate that they belong to the same consumer group.
<code>batchSize</code>	<code>1000</code>	The maximum number of messages that can be written to a channel in a single batch.
<code>batchDurationMillis</code>	<code>1000</code>	The maximum time (in ms) before a batch is written to the channel. The batch is written when the <code>batchSize</code> limit or <code>batchDurationMillis</code> 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 <code>kafka.</code> (for example, <code>kafka.fetch.min.bytes</code>). See the Kafka documentation for the full list of Kafka consumer properties.

Tuning Notes

The Kafka source overrides two Kafka consumer parameters:

- `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.
- `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 <code>org.apache.flume.sink.kafka.KafkaSink</code> .
brokerList		The brokers the Kafka sink uses to discover topic partitions, formatted as a comma-separated list of <code>hostname:port</code> 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 <code>topic</code> 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 <code>batchSize</code> 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 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 <code>kafka.</code> (for example, <code>kafka.compression.codec</code>). 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 <code>org.apache.flume.channel.kafka.KafkaChannel</code> .
brokerList		The brokers the Kafka channel uses to discover topic partitions, formatted as a comma-separated list of <code>hostname:port</code> 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 host (for example, <code>zk01.example.com:2181</code>) or a comma-separated list of hosts in a ZooKeeper quorum (for example, <code>zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181</code>).
topic	<code>flume-channel</code>	The Kafka topic the channel will use.
groupID	<code>flume</code>	The unique identifier of the Kafka consumer group the channel uses to register with Kafka.
parseAsFlumeEvent	<code>true</code>	Set to <code>true</code> if a Flume source is writing to the channel and expects AvroDataums with the FlumeEvent schema (<code>org.apache.flume.source.avro.AvroFlumeEvent</code>) in the channel. Set to <code>false</code> if other producers are writing to the topic that the channel is using.

Property Name	Default Value	Description
readSmallestOffset	false	If <code>true</code> , reads all data in the topic. If <code>false</code> , reads only data written after the channel has started. Only used when <code>parseAsFlumeEvent</code> is <code>false</code> .
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 <code>kafka.</code> (for example, <code>kafka.compression.codec</code>). See the Kafka documentation for the full list of Kafka producer properties.

Additional Information

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
```

See the [official Kafka documentation](#).

See [Kafka FAQ](#).

Kafka Administration

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

For a complete listing of available configuration settings, see [Kafka Properties in CDH 5.4.0](#).

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 one. For high availability production systems, Cloudera recommends setting the replication factor to at least three. This requires at least three 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.

Preferred Leader Election

Kafka is designed with failure in mind. At some point in time, web communications or storage resources fail. When a broker goes offline, one of the replicas becomes the new leader for the partition. When the broker comes back online, it has no leader partitions. Kafka keeps track of which machine is configured to be the leader. Once the original broker is back up and in a good state, Kafka restores the information it missed in the interim and makes it the partition leader once more.

Preferred Leader Election is enabled by default, and should occur automatically unless you actively disable the feature. Typically, the leader is restored within five minutes of coming back online. If the preferred leader is offline for a very long time, though, it might need additional time to restore its required information from the replica.

There is a small possibility that some messages might be lost when switching back to the preferred leader. You can minimize the chance of lost data by setting the `kafka.request.required.acks` property on the Producer to `-1`. See [Acknowledgements](#) on page 24.

Unclean Leader Election

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. 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.

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 `request.required.acks` property (see [Table 3: Kafka Sink Properties](#) on page 21 for details).

Set `request.required.acks` 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 `request.required.acks` to `-1` provides the highest consistency guarantee at the expense of slower writes to the cluster.

Minimum In-sync Replicas

You can set the minimum number of in-sync replicas (ISRs) 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 `request.required.acks` 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 hosts crashes, the partition is no longer available for writes. Similar 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).

The leader is considered one of the in-sync replicas. It is included in the count of total `min.insync.replicas`. However, leaders are special, in that producers and consumers can only interact with leaders in a Kafka cluster.

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. You can configure MirrorMaker directly in Cloudera Manager 5.4 and higher.

The most important configuration setting is **Destination broker list**. This is a list of brokers on the destination cluster. You should list more than one, to support high availability, but you do not need to list all brokers.

You can create a **Topic whitelist** that represents the exclusive set of topics to replicate. Alternatively, you can define a **Topic blacklist** that represents the list of topics that should not be replicated. These values are mutually exclusive: if both are defined, Kafka uses the whitelist settings.

For production use, select the **Avoid data loss** setting. 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 received since its previous checkpoint.

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. With larger messages, you might need to use fewer partitions or provide more RAM.

Partition Reassignment

At some point you will likely exceed configured resources on your system. If you add a Kafka broker to your cluster to handle increased demand, new partitions are allocated to it (the same as any other broker), but it does not automatically share the load of existing partitions on other brokers. To redistribute the existing load among brokers, you must manually reassign partitions. You can do so using `bin/kafka-reassign-partitions.sh` script utilities.

To reassign partitions:

1. Create a list of topics you want to move.

```
topics-to-move.json
{"topics": [{"topic": "foo1"},
             {"topic": "foo2"}],
  "version":1
}
```

2. Use the `--generate` option in `kafka-reassign-partitions.sh` to list the distribution of partitions and replicas on your current brokers, followed by a list of suggested locations for partitions on your new broker.

```
bin/kafka-reassign-partitions.sh --zookeeper localhost:2181
  --topics-to-move-json-file topics-to-move.json
  --broker-list "4"
  --generate

Current partition replica assignment

{"version":1,
 "partitions":[{"topic":"foo1","partition":2,"replicas":[1,2]},
               {"topic":"foo1","partition":0,"replicas":[3,1]},
               {"topic":"foo2","partition":2,"replicas":[1,2]},
               {"topic":"foo2","partition":0,"replicas":[3,2]},
               {"topic":"foo1","partition":1,"replicas":[2,3]},
               {"topic":"foo2","partition":1,"replicas":[2,3]}]
}

{"version":1,
 "partitions":[{"topic":"foo1","partition":3,"replicas":4},
               {"topic":"foo1","partition":1,"replicas":4},
               {"topic":"foo2","partition":2,"replicas":4}]
}
```

3. Revise the suggested list if required, and then save it as a JSON file.
4. Use the `--execute` option in `kafka-reassign-partitions.sh` to start the redistribution process, which can take several hours in some cases.

```
> bin/kafka-reassign-partitions.sh \
  --zookeeper localhost:2181 \
  --reassignment-json-file expand-cluster-reassignment.json
  --execute
```

5. Use the `--verify` option in `kafka-reassign-partitions.sh` to check the status of your partitions.

Although reassigning partitions is labor-intensive, you should anticipate system growth and redistribute the load when your system is at 70% capacity. If you wait until you are forced to redistribute because you have reached the limits of your resources, the redistribution process can be extremely slow.

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

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.codec` and `compressed.topics` producer configuration parameters to enable compression. 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.segment.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)

Tuning Kafka for Optimal Performance

Performance tuning involves two important metrics: *Latency* measures how long it takes to process one event, and *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.

Tuning Kafka Producers

Kafka uses an asynchronous publish/subscribe model. When your producer calls the `send()` command, 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 lead time.

Batch Size

`send.buffer.bytes` 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.

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 wait 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.

Lead Time

`queue.buffering.max.ms` sets the maximum time to buffer data in asynchronous mode. For example, a setting of 100 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 `queue.buffering.max.ms` to 5 and send a more messages in one batch.

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

Tuning Kafka Brokers

Topics are divided into partitions. Each partition has a leader. Most partitions are written into leaders with multiple replicas. When the leaders are not balanced properly, one might be overworked, compared to others. For more information on load balancing, see [Partitions and Memory Usage](#).

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. Cloudera recommends starting with one partition per physical storage disk and one consumer per partition.

Tuning Kafka Consumers

Consumers can create throughput issues on the other side of the pipeline. The maximum number of consumers 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. Adding more consumer groups does not affect performance.


How you use the `replica.high.watermark.checkpoint.interval.ms` property can affect throughput. When reading from a partition, you can mark the last point where you read information. That way, if you have to go back and locate missing data, you have a checkpoint from which to move forward without having to reread prior data. If you set the checkpoint watermark for every event, you will never lose a message, but it significantly impacts performance. If, instead, you set it to check the offset every hundred messages, you have a margin of safety with much less impact on throughput.

Metrics

This section lists examples of the most commonly used metrics, their significance, and configuration changes to consider in response to metric variations. For the complete list, see [Kafka Metrics](#).

Kafka Cluster Metrics

Metric	Description	Significance	Action
Active Controllers	Shows a line for each broker that acted as an active controller during the charted time period.	A non-zero value indicates that the broker was the active controller during that time. When zoomed out to non-raw data, fractional values can occur during transitions between active controllers.	Some issues, such as failure of the <i>Create Topic</i> command, require that you check controller logs. Check the Active Controllers metric to see which broker was the controller when the issue occurred.
Total Messages Received Across Kafka Brokers	Number of messages received from producers.	This is an indicator of overall workload, based on the quantity of messages.	Consider adding resources when workload approaches maximum capacity.
Total Bytes Received Across Kafka Brokers	Amount of data broker received from producers.	This is an indicator of overall workload, based on the size of messages.	Consider adding resources when workload approaches maximum capacity.
Total Bytes Fetched Across Kafka Brokers	Amount of data consumers read from broker.	This is an indicator of overall workload, based on consumer demand.	Consider adding resources when workload approaches maximum capacity.
Total Partitions Across Kafka Brokers	Number of partitions (lead or follower replicas) on broker.	Cloudera does not recommend more than 2000 partitions per broker.	Consider adding additional brokers and rebalance partitions.
Total Leader Replicas Across Kafka Brokers	Number of leader replicas on broker.	Total Leader Replicas should be roughly the same in all brokers. If one broker has significantly more Lead Replicas, it might be overloaded (check network, cpu and disk metrics to see if this is the case).	Set Enable automatic rebalancing of leadership to preferred replicas to true .
Total Offline Partitions Across Kafka Brokers	The number of unavailable partitions.	Offline partitions are not available for reading and writing. This can happen for several reasons (for example, when brokers for all available partitions are down).	Restart the brokers, if needed, and check the logs for errors.
Total Under Replicated Partitions Across Kafka Brokers	The number of partitions with unavailable replicas.	Under-replicated partitions means that one or more replicas are not available. This is usually because a broker is down.	Restart the broker, and check for errors in the logs.

Metric	Description	Significance	Action
Informational Events	The number of informational events.	An event is a record that something of interest has occurred – a service's health has changed state, a log message (of the appropriate severity) has been logged, and so on. Many events are enabled and configured by default. See Events .	See Configuring Monitoring Settings .
Important Events and Alerts	The number of recent alerts and important or critical events.	An alert is an event that is considered especially noteworthy and is triggered by a selected event. Alerts are shown with an  badge when they appear in a list of events. You can configure the Alert Publisher to send alert notifications by email or via SNMP trap to a trap receiver. See Alerts .	See Managing Alerts .

Kafka Broker Metrics in Cloudera Manager

These metrics are tracked by default. You can add some or all of these metrics to the standard dashboard, or create a custom dashboard with only those items of particular interest. All of the metrics you can see at cluster level can also be shown at broker level.

Metric	Description	Significance	Action
Health	The percentage of time this entity has spent in various health states. This chart can be used to see times in the past when the entity was healthy or unhealthy and to get a visual representation of the amount of time it was healthy or unhealthy.	Checks the amount of swap memory in use by the role. A failure of this health test might indicate that your machine is overloaded.	Adjust Process Swap Memory Thresholds monitoring settings for this role instance.
Host Memory Usage	Host memory usage, broken into various usage categories, including <i>swap</i> .	The host's memory capacity is shown as a horizontal line near the top of the chart. An overcommitted host's usage extends past this line.	Adjust Process Swap Memory Thresholds for this role instance.
Host Swap Rate	Host memory/disk swap rate.	In general, any swap is undesirable. Non-trivial swapping can lead to performance issues.	Adjust Process Swap Memory Thresholds for this role instance.
Host CPU Usage	Host CPU usage, broken into user and system usage.		Adjust Cgroup CPU Shares for this Host instance.

Metric	Description	Significance	Action
Role CPU Usage	Role CPU usage, broken into user and system usage.		Adjust Cgroup CPU Shares for this role instance.
Resident Memory	Resident memory in use.		Set Cgroup Memory Soft Limit and Cgroup Memory Hard Limit to <code>-1</code> to specify there is no limit. Consider adding resources of the cluster.
Host Network Throughput	The total network read and write I/O, across all of the host's network interfaces.		Consider adding resources to the host, or move partitions to a different broker.
Disk Latency	Latency statistics across each of the host's interfaces.		Consider adding resources to the host, or move partitions to a different broker.
Aggregate Disk Throughput	Total disk read and write I/O, across all of the host's disks.		Consider adding resources to the host, or move partitions to a different broker.
Aggregate Disk IOPS	Total disk read and write IOPS, across all of the host's disks.		Consider adding resources to the host, or move partitions to a different broker.
ISR Expansions	Number of times In-Sync Replicas for a partition expanded.	If a broker goes down, ISR for some of the partitions shrink. When that broker is up again, ISRs are expanded once the replicas are fully caught up. Other than that, the expected value for ISR expansion rate is 0.	If ISR is expanding and shrinking frequently, adjust Allowed replica lag .
ISR Shrinks	Number of times In-Sync Replicas for a partition shrank.	If a broker goes down, ISR for some of the partitions shrink. When that broker is up again, ISRs are expanded once the replicas are fully caught up. Other than that, the expected value for ISR shrink rate is 0.	If ISR is expanding and shrinking frequently, adjust Allowed replica lag .

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.

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.
```