cloudera

Cloudera Distribution of Apache 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.
1001 Page Mill Road, Bldg 3
Palo Alto, CA 94304
info@cloudera.com
US: 1-888-789-1488
Intl: 1-650-362-0488
www.cloudera.com

Release Information

Version: Cloudera Distribution of Apache Kafka 2.2.x
Date: July 10, 2017

Table of Contents

Cloudera Distribution of Apache Kafka Overview

Cloudera Distribution of Apache Kafka is a distributed commit log service. Kafka functions much like a publish/subscribe messaging system, but with better throughput, built-in partitioning, replication, and fault tolerance. Kafka is a good solution for large scale message processing applications. It is often used in tandem with Apache Hadoop, Apache Storm, and Spark Streaming.

You might think of a log as a time-sorted file or data table. Newer entries are appended to the log over time, from left to right. The log entry number is a convenient replacement for a timestamp.

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 original use case for Kafka was to track user behavior on websites. Site activity (page views, searches, or other actions users might take) is published to central topics, with one topic per activity type.

Kafka can be used to monitor operational data, aggregating statistics from distributed applications to produce centralized data feeds. It also works well for log aggregation, with low latency and convenient support for multiple data sources.

Kafka provides the following:

- Persistent messaging with O(1) disk structures, meaning that the execution time of Kafka's algorithms is independent of the size of the input. Execution time is constant, 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. It distributes consumption over a cluster of consumer machines while maintaining the order of the message stream.
- Support for parallel data load into Hadoop.

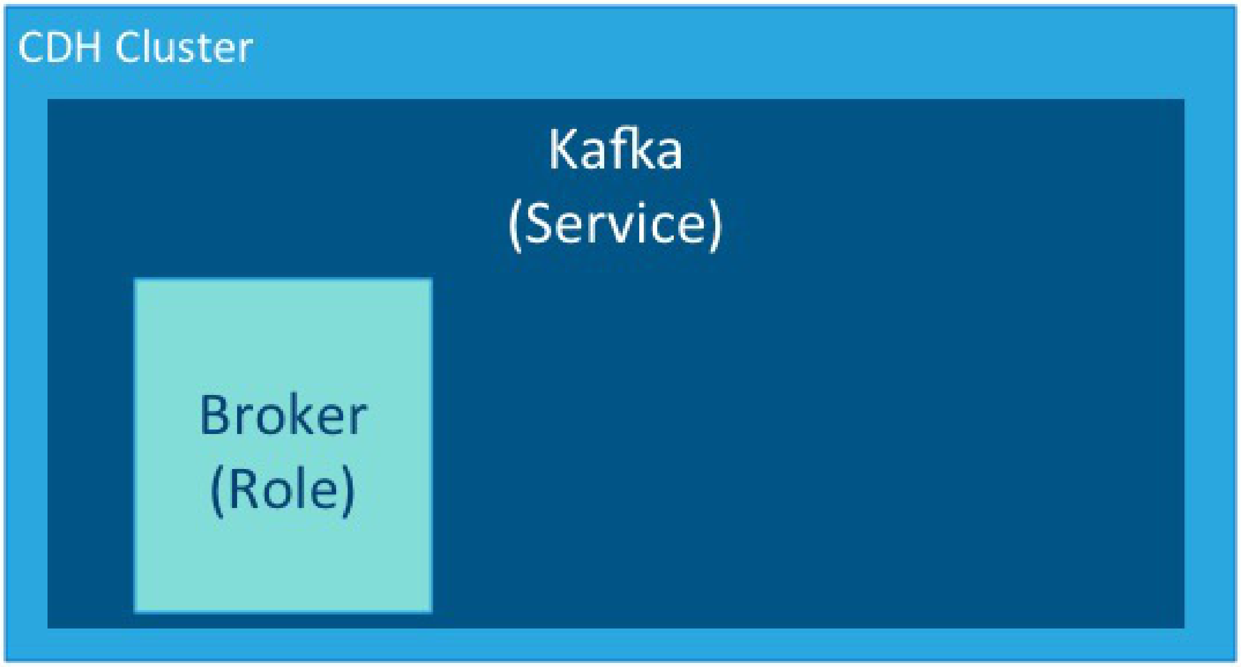
Understanding Kafka Terminology

Kafka and Cloudera Manager use terms in ways that might vary from other technologies. This topic provides definitions for how these terms are used in Kafka with Cloudera Manager.

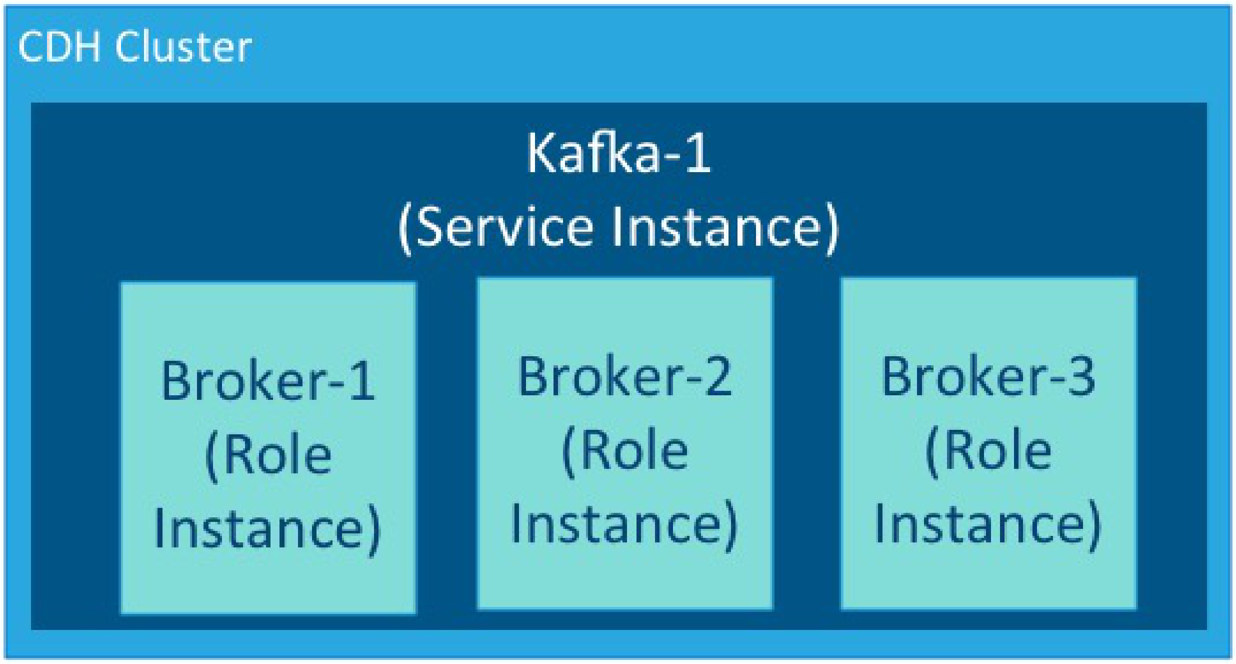
For a quick video overview of the terms, see tl;dr:Talk Like Kafka

A *service* is an application that runs in a CDH cluster. Kafka is a service. ZooKeeper is a service that runs within a Kafka cluster. Other services include MapReduce, HDFS, YARN, Flume, and Spark.

A *role* is a feature of a service. A *broker* is a role in a Kafka service.

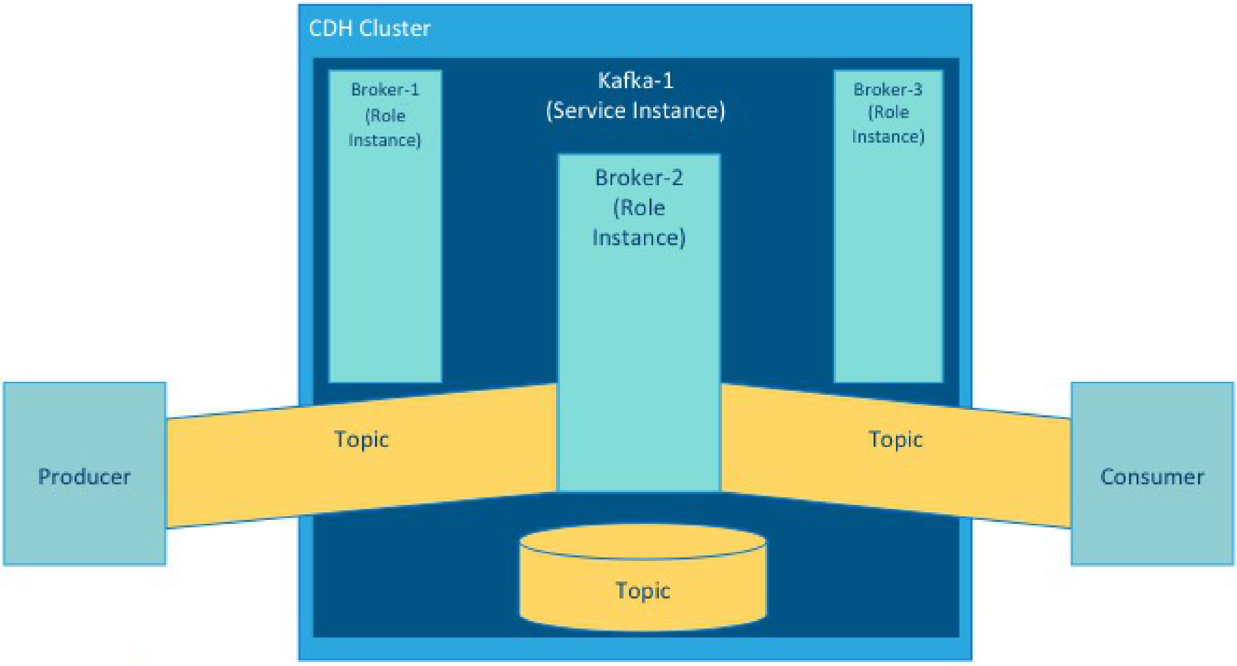


An *instance* is a deployed and configured software component. A cluster can include multiple roles and multiple instances of the same role. A *service instance* might be `Kafka-1.` `Kafka-1` might host the *role instances* `Broker-1`, `Broker-2`, and `Broker-3`.

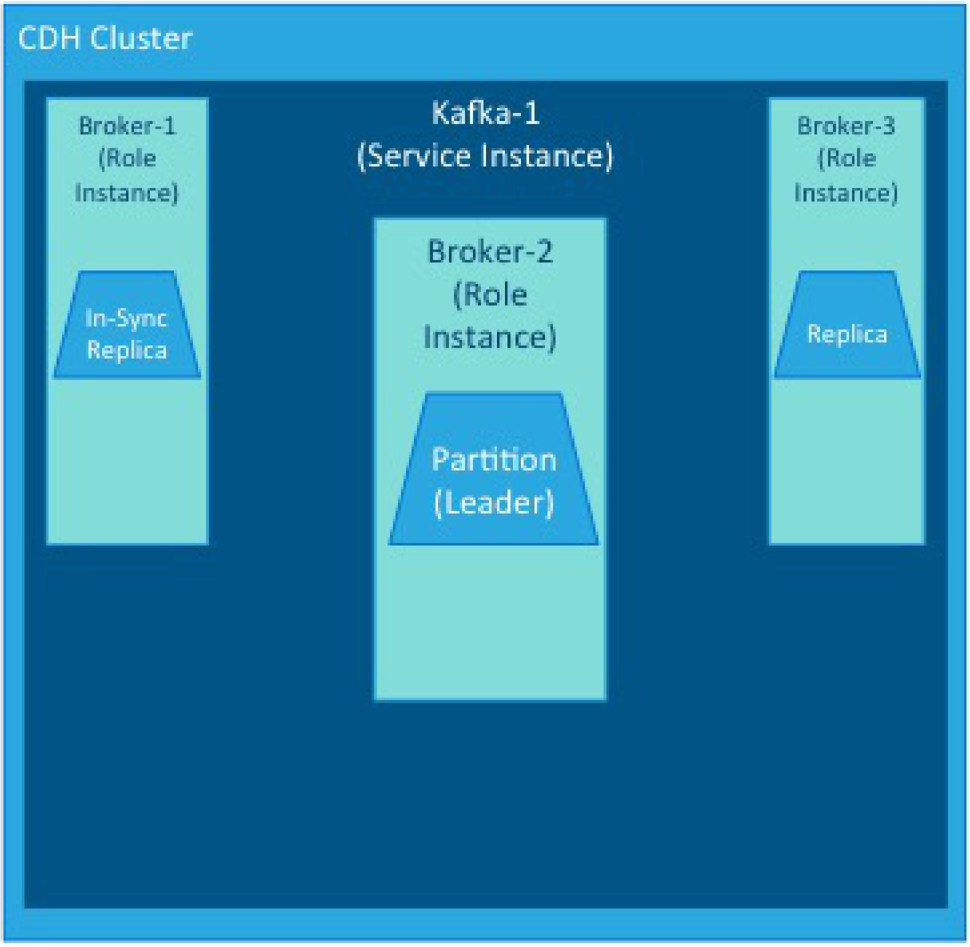


Kafka brokers process *records* organized into *topics*. A topic is a category of records that share similar characteristics. For example, a topic might consist of instant messages from social media or navigation information for users on a web site. Each topic has a unique corresponding table in data storage.

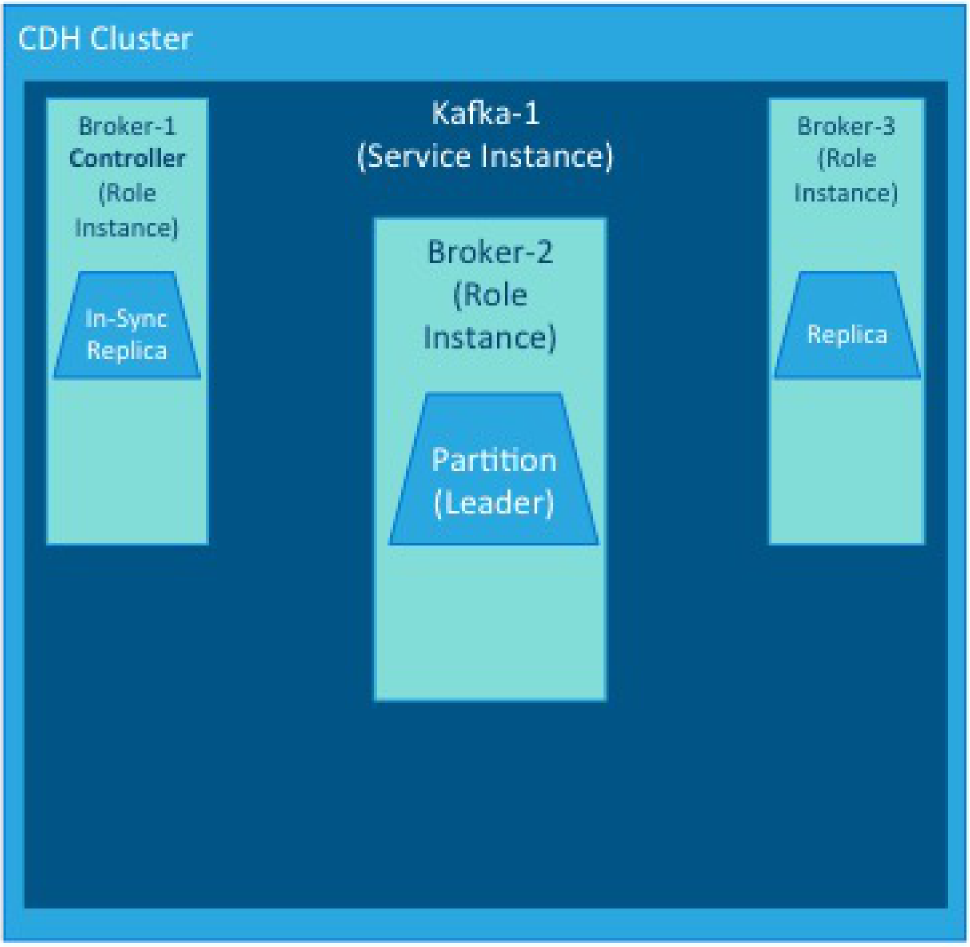
A *producer* is an external process that sends records to a Kafka topic. A *consumer* is an external process that receives topic streams from a Kafka cluster.



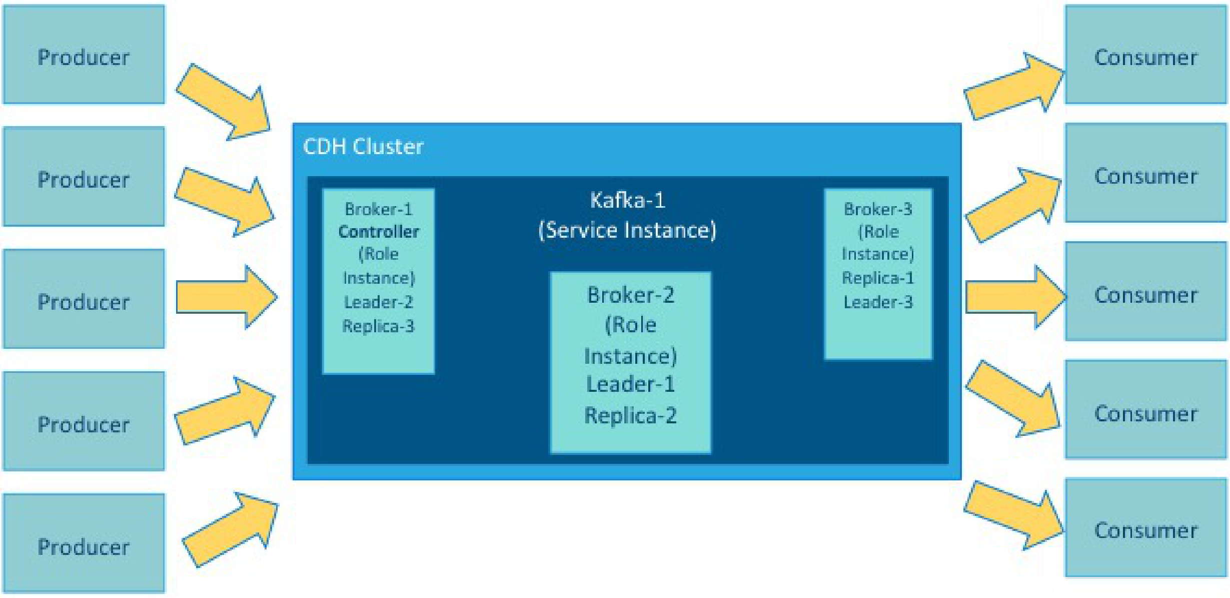
Brokers process topics in *partitions*. A partition on one broker in a cluster is the *leader*. The same partition is mirrored on one or more other brokers in the cluster as *replicas*. When a leader goes offline, a replica automatically takes its place and becomes the new leader for the topic. An *in-sync replica* is a replica that is completely up-to-date with the leader.



Each Kafka cluster has one broker that also acts as the *controller*. The controller is responsible for managing the states of partitions and replicas. It also performs administrative tasks, such as reassigning partitions.



While these illustrations show single instances of the components of a Kafka implementation, Kafka brokers typically host multiple partitions and replicas, with any number of producers and consumers, up to the requirements and limits of the deployed system.



Cloudera Distribution of Apache Kafka Release Notes

Cloudera Distribution of Apache Kafka provides a release guide that contains release and download information for installers and administrators. It includes release notes as well as information about versions and downloads. The guide also provides a release matrix that shows which major and minor release version of a product is supported with which release version of Cloudera Distribution of Apache Kafka.

The Release Guide is comprised of topics including:

What's New in Cloudera Distribution of Apache Kafka?

This section lists new features in Cloudera Distribution of Apache Kafka. The following links provide detailed information for each release:

New Features in Cloudera Distribution of Apache Kafka 2.2.0

- **Rebase on Kafka 0.10.2**

  Cloudera Distribution of Apache Kafka 2.2.0 is rebased on Apache Kafka 0.10.2. See https://www.apache.org/dyn/closer.cgi?path=/kafka/0.10.2.0/RELEASE_NOTES.html .

New Features in Cloudera Distribution of Apache Kafka 2.1.0

- **Rebase on Kafka 0.10**

  Cloudera Distribution of Apache Kafka 2.1.0 is rebased on Apache Kafka 0.10. See https://www.apache.org/dyn/closer.cgi?path=/kafka/0.10.0.0/RELEASE_NOTES.html .

- **Sentry Authentication**

  Apache Sentry includes Kafka binding you can use to enable authorization in Kafka with Sentry. See Configuring Kafka to Use Sentry Authorization on page 42.

New Features in Cloudera Distribution of Apache Kafka 2.0.0

- **Rebase on Kafka 0.9**

  Cloudera Distribution of Apache Kafka 2.0.0 is rebased on Apache Kafka 0.9. See https://www.apache.org/dyn/closer.cgi?path=/kafka/0.9.0.0/RELEASE_NOTES.html.

- **Kerberos**

  Cloudera Distribution of Apache Kafka 2.0.0 supports Kerberos authentication of connections from clients and other brokers, including to ZooKeeper.

- **SSL**

  Cloudera Distribution of Apache Kafka 2.0.0 supports wire encryption of communications from clients and other brokers using SSL.

- **New Consumer API**

  Cloudera Distribution of Apache Kafka 2.0.0 includes a new Java API for consumers.

- **MirrorMaker**

  MirrorMaker is enhanced to help prevent data loss and improve reliability of cross-data center replication.

- **Quotas**

You can use per-user quotas to throttle producer and consumer throughput in a multitenant cluster. See Quotas on page 50.

New Features in Cloudera Distribution of Apache Kafka 1.4.0

- Cloudera Distribution of Apache 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 21.

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 12

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 Reviewing and Responding to Changes in Kafka Metrics on page 51.

- **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 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.

Cloudera Distribution of Apache Kafka Requirements and Supported Versions

The following sections describe software requirements and supported versions of complementary software for Cloudera Distribution of Apache Kafka:

Supported CDH and Cloudera Manager Releases

The following table shows the suported releases of CDH and Cloudera Manager for Kafka releases.

Kafka Release	CDH Release	Cloudera Manager Release
Kafka 2.2.x	CDH 5.4.x and higher	Cloudera Manager 5.5.3 and higher
Kafka 2.1.x	CDH 5.4.x and higher	Cloudera Manager 5.5.3 and higher
Kafka 2.0.x	CDH 5.4.x and higher	Cloudera Manager 5.5.3 and higher
Kafka 1.x	CDH 5.4.x, 5.5x, 5.6.x	Cloudera Manager 5.4.x and higher

Supported Integrations

- Flume and Spark connectors to Kafka are included with CDH 5.7.x and higher and only work with Kafka 2.0.x and higher.
- Sentry authorization integration with Kafka only works with Kafka 2.1.x and higher and CDH 5.9.x and higher.

Supported Operating Systems

Cloudera Distribution of Apache Kafka 2.2.x 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
	5.11	64-bit
	6.4	64-bit
	6.5	64-bit
	6.5 in SE Linux mode	64-bit
	6.6	64-bit
	6.7	64-bit
	6.9	64-bit
	7.1	64-bit
	7.2	64-bit
	7.3	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

Operating System	Version	Packages
	6.6	64-bit
	6.6 in SE Linux mode	64-bit
	6.7	64-bit
	6.9	64-bit
	7.1	64-bit
	7.2	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
	7.2	64-bit
SLES		
SUSE Linux Enterprise Server (SLES)	11 with Service Pack 2	64-bit
	11 with Service Pack 3	64-bit
	11 with Service Pack 4	64-bit
	12 with Service Pack 1	64-bit
Ubuntu/Debian		
Ubuntu	Precise (12.04) - Long-Term Support (LTS)	64-bit
	Trusty (14.04) - Long-Term Support (LTS)	64-bit
	Xenial (16.04) - Long-Term Support (LTS)	64-bit
Debian	Wheezy (7.8, 7.0, 7.1), Jessie (8.2, 8.4)	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:** The minimum supported and recommended JDK versions include one exception: 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.

> **Note:** If you decide to use the G1 garbage collector and you use JDK 1.7, make sure you use u51 or newer.

Ports Used by Kafka

Kafka uses the TCP ports listed in the following table. Before deploying Kafka, ensure that these ports are open on each system.

Component	Service	Port	Access Requirement	Comment
Broker	TCP Port	9092	External/Internal	The primary communication port used by producers and consumers; also used for inter-broker communication.
Broker	TLS/SSL Port	9093	External/Internal	A secured communication port used by producers and consumers; also used for inter-broker communication.
Broker	JMX Port	9393	Internal	Internal use only. Used for administration via JMX.
MirrorMaker	JMX Port	9394	Internal	Internal use only. Used to administer the producer and consumer of the MirrorMaker.
Broker	HTTP Metric Report Port	24042	Internal	Internal use only. This is the port via which the HTTP metric reporter listens. It is used to retrieve metrics through HTTP instead of JMX.

Issues Fixed in Cloudera Distribution of Apache Kafka

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

Issues Fixed in Cloudera Distribution of Apache Kafka 2.2.0

- KAFKA-4525; Kafka should not require SSL truststore password

- KAFKA-4811; ReplicaFetchThread may fail to create due to existing metric
- KAFKA-4741; Fix potential buffer leak in RecordAccumulator in case of exception
- KAFKA-4735; Fix deadlock issue during MM shutdown
- KAFKA-4636; Per listener security settings overrides (KIP-103)
- KAFKA-5150; Reduce lz4 decompression overhead
- KAFKA-5316; LogCleaner should account for larger record sets after cleaning
- KAFKA-5097; Fix regression in consumer caused by unsafe access to potentially unassigned partitions
- KAFKA-4959; Remove controller concurrent access to non-threadsafe NetworkClient, Selector, and SSLEngine
- KAFKA-4631; Request metadata in consumer if topic/partitions unavailable

Issues Fixed in Cloudera Distribution of Apache Kafka 2.1.1

- KAFKA-724: Allow automatic socket.send.buffer from operating system in SocketServer
- KAFKA-2684: Add force option to topic / config command so they can be called programatically
- KAFKA-2720: Expire group metadata when all offsets have expired
- KAFKA-2948: Remove unused topics from producer metadata set
- KAFKA-3111: Fix ConsumerPerformance reporting to use time-based instead of message-based intervals
- KAFKA-3158: ConsumerGroupCommand should tell whether group is actually dead
- KAFKA-3175: Topic not accessible after deletion even when delete.topic.enable is disabled
- KAFKA-3501: Console consumer process hangs on exit
- KAFKA-3562: Handle topic deletion during a send
- KAFKA-3645: Fix ConsumerGroupCommand and ConsumerOffsetChecker to correctly read endpoint info from ZK
- KAFKA-3716: Validate all timestamps are not negative
- KAFKA-3719: Allow underscores in hostname
- KAFKA-3748: Add consumer-property to console tools consumer
- KAFKA-3774: Make 'time' an optional argument of GetOffsetShell
- KAFKA-3810: replication of internal topics should not be limited by replica.fetch.max.bytes
- KAFKA-3934: Start scripts enable GC by default with no way to disable
- KAFKA-3965: MirrorMaker should not commit offset when exception is thrown from producer.send
- KAFKA-4158: Reset quota to default value if quota override is deleted
- KAFKA-4229: Controller can't start after several zk expired event
- KAFKA-4319: Parallelize shutdown of fetchers in AbstractFetcherManager to speedup shutdown
- KAFKA-4360: Controller may deadLock when autoLeaderRebalance encounter ZK expired
- KAFKA-4428: Kafka does not exit if port is already bound

Issues Fixed in Cloudera Distribution of Apache Kafka 2.1.0

- KAFKA-3787: Preserve the message timestamp in MirrorMaker
- KAFKA-3789: Upgrade Snappy to fix Snappy decompression errors
- KAFKA-3802: Log mtimes reset on broker restart or shutdown
- KAFKA-3894: Log cleaner can partially clean a segment
- KAFKA-3915: Do not convert messages from v0 to v1 during log compaction
- KAFKA-3933: Always fully read deepIterator
- KAFKA-3950: Only throw authorization exception if pattern subscription matches topic
- KAFKA-3977: Defer fetch parsing for space efficiency and to ensure exceptions are raised to the user
- KAFKA-4050: Allow configuration of the PRNG used for SSL
- KAFKA-4073: MirrorMaker should handle messages without timestamp correctly

Cloudera Distribution of Apache Kafka 2.1.0 is rebased on Apache Kafka 0.10. For a complete list of fixed issues, see https://www.apache.org/dyn/closer.cgi?path=/kafka/0.10.0.0/RELEASE_NOTES.html.

Issues Fixed in Cloudera Distribution of Apache Kafka 2.0.2

- KAFKA-3495: `NetworkClient.blockingSendAndReceive` should rely on `requestTimeout`.
- KAFKA-2998: Log warnings when client is disconnected from bootstrap brokers.
- KAFKA-3488: Avoid failing of unsent requests in consumer where possible.
- KAFKA-3528: Handle wakeups while rebalancing more gracefully.
- KAFKA-3594: After calling `MemoryRecords.close()` method, `hasRoomFor()` method should return `false`.
- KAFKA-3602: Rename RecordAccumulator `dequeFor()` and ensure proper usage.
- KAFKA-3789: Upgrade Snappy to fix Snappy decompression errors.
- KAFKA-3830: `getTGT()` debug logging exposes confidential information.
- KAFKA-3840: Allow clients default OS buffer sizes.
- KAFKA-3691: Confusing logging during metadata update timeout.
- KAFKA-3810: Replication of internal topics should not be limited by `replica.fetch.max.bytes`.
- KAFKA-3854: Fix issues with new consumer's subsequent regex (pattern) subscriptions.

Issues Fixed in Cloudera Distribution of Apache Kafka 2.0.1

- KAFKA-3409: MirrorMaker hangs indefinitely due to commit.
- KAFKA-3378: Client blocks forever if SocketChannel connects instantly.
- KAFKA-3426: Improve protocol type errors when invalid sizes are received.
- KAFKA-3330: Truncate log cleaner offset checkpoint if the log is truncated.
- KAFKA-3463: Change default receive buffer size for consumer to 64K.
- KAFKA-1148: Delayed fetch/producer requests should be satisfied on a leader change.
- KAFKA-3352: Avoid DNS reverse lookups.
- KAFKA-3341: Improve error handling on invalid requests.
- KAFKA-3310: Fix for NPEs observed when throttling clients.
- KAFKA-2784: swallow exceptions when MirrorMaker exits.
- KAFKA-3243: Fix Kafka basic ops documentation for MirrorMaker, blacklist is not supported for new consumers.
- KAFKA-3235: Unclosed stream in AppInfoParser static block.
- KAFKA-3147: Memory records is not writable in MirrorMaker.
- KAFKA-3088: Broker crash on receipt of produce request with empty client ID.
- KAFKA-3159: Kafka consumer client poll is very CPU intensive under certain conditions.
- KAFKA-3189: Kafka server returns UnknownServerException for inherited exceptions.
- KAFKA-3157: MirrorMaker does not commit offset with low traffic.
- KAFKA-3179: Kafka consumer delivers message whose offset is earlier than sought offset.
- KAFKA-3198: Ticket Renewal Thread exits prematurely due to inverted comparison.

Issues Fixed in Cloudera Distribution of Apache Kafka 2.0.0

- KAFKA-2799: WakeupException thrown in the followup poll() could lead to data loss.
- KAFKA-2878: Kafka broker throws OutOfMemory exception with invalid join group request.
- KAFKA-2880: Fetcher.getTopicMetadata NullPointerException when broker cannot be reached.
- KAFKA-2882: Add constructor cache for Snappy and LZ4 Output/Input streams in Compressor.java
- KAFKA-2913: GroupMetadataManager unloads all groups in removeGroupsForPartitions.
- KAFKA-2942: Inadvertent auto-commit when pre-fetching can cause message loss.
- KAFKA-2950: Fix performance regression in the producer.
- KAFKA-2973: Fix leak of child sensors on remove.
- KAFKA-2978: Consumer stops fetching when consumed and fetch positions get out of sync.
- KAFKA-2988: Change default configuration of the log cleaner.
- KAFKA-3012: Avoid reserved.broker.max.id collisions on upgrade.

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 '.'

Known Issues in Cloudera Distribution of Apache Kafka

The following sections describe known issues in Cloudera Distribution of Apache Kafka:

Kafka client jars included in CDH may not match the newest Kafka parcel jar

The Kafka client jars included in CDH may not match the newest Kafka parcel jar that is released. This is done to maintain compatibility across CDH 5.7 and higher for integrations such as Spark and Flume.

Source cluster not definable in Kafka 1.x

In Kafka 1.x, the source cluster is assumed to be the cluster that MirrorMaker is running on. In Kafka 2.0, you can define a custom source and target cluster.

Kafka fails when configured with Sentry and an old Kafka version.

If Kafka is configured with Sentry, and Kafka is not of version 2.1 or higher, Kafka fails with `java.lang.ClassNotFoundException: org.apache.sentry.kafka.authorizer.SentryKafkaAuthorizer`

Workaround: unset the Kafka to Sentry dependency.

The Flume and Spark connectors to Kafka shipped with CDH 5.7 and higher only work with Kafka 2.0 and higher.

Use Kafka 2.0 and higher to be compatible with the Flume and Spark connectors included with CDH 5.7.x.

Only new Java clients support authentication and authorization

Workaround: Migrate to the new Java producer and consumer APIs.

Requests fail when sending to a nonexistent topic with `auto.create.topics.enable` set to `true`

The first few produce requests fail when sending to a nonexistent topic with `auto.create.topics.enable` set to `true`.

Workaround: Increase the number of `retries` in the Producer configuration settings.

Custom Kerberos principal names must not be used for Kerberized ZooKeeper and Kafka instances

When using ZooKeeper authentication and a custom Kerberos principal, Kerberos-enabled Kafka does not start.

Workaround: None. You must disable ZooKeeper authentication for Kafka or use the default Kerberos principals for ZooKeeper and Kafka.

Performance degradation when SSL is enabled

Significant performance degradation can occur when SSL is enabled. The impact varies, depending on your CPU type and JVM version. The reduction is generally in the range 20-50%.

Unsupported features

- Kafka Connect is included with Cloudera Distribution of Apache Kafka 2.0.0, but is not supported at this time.
- The Kafka default authorizer is included with Cloudera Distribution of Apache Kafka 2.0.0, but is not supported at this time. This includes setting ACLs and all related APIs, broker functionality, and command-line tools.

AdminUtils is not binary-compatible between Cloudera Distribution of Apache Kafka 1.x and 2.x

The AdminUtils APIs have changed between Cloudera Distribution of Apache Kafka 1.x and 2.x. If your application uses AdminUtils APIs, you must modify your application code to use the new APIs before you compile your application against Cloudera Distribution of Apache Kafka 2.x.

> **Note:** AdminUtils APIs are not part of the publicly supported Cloudera Distribution of Apache Kafka API.

Monitoring is not supported in Cloudera Manager 5.4

If you use Cloudera Distribution of Kafka 1.2 with Cloudera Manager 5.4, you must disable monitoring.

Cloudera Distribution of Apache Kafka Incompatible Changes and Limitations

This section describes incompatible changes and limitations:

> **Warning:** The open file handlers of 2.2.0 will increase by roughly 33% because of the addition of time index files for each segment.

Flume shipped with CDH 5.6 and higher can only send data to Kafka 2.0 and higher via unsecured transport.

Security additions to Kafka 2.0 are not supported by CDH.

Topic Blacklist Removed

The MirrorMaker **Topic blacklist** setting has been removed in Cloudera Distribution of Kafka 2.0 and higher.

Avoid Data Loss Option Removed

The **Avoid Data Loss** option from earlier releases has been removed in Kafka 2.0 in favor of automatically setting the following properties.

1. Producer settings
   - `acks=all`
   - `retries=max integer`
   - `max.block.ms=max long`
2. Consumer setting
   - `auto.commit.enable=false`
3. MirrorMaker setting
   - `abort.on.send.failute=true`

Cloudera Distribution of Apache Kafka Version and Packaging Information

This section describes naming conventions for Cloudera Distribution of Apache Kafka package versions, lists versions and where to download components.

For installation instructions, see Installing, Migrating and Upgrading Kafka on page 25.

Examples of Cloudera Distribution of Apache Kafka 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	Parcel Repository
2.2.0	Apache Kafka	0.10.2.0+kafka2.2.0+110	Release notes	Cloudera Distribution of Apache Kafka 2.2.0 Parcel Repository
2.1.1	Apache Kafka	0.10.0.0+kafka2.1.1+21	Release notes	Cloudera Distribution of Apache Kafka 2.1.1 Parcel Repository
2.1.0	Apache Kafka	0.10.0.0+kafka2.1.0+63	Release notes	Cloudera Distribution of Apache Kafka 2.1.0 Parcel Repository
2.0.2	Apache Kafka	0.9.0.0+kafka2.0.2+305	Release notes	Cloudera Distribution of Apache Kafka 2.0.2 Parcel Repository
2.0.1	Apache Kafka	0.9.0.0+kafka2.0.1+283	Release notes	Cloudera Distribution of Apache Kafka 2.0.1 Parcel Repository
2.0.0	Apache Kafka	0.9.0.0+kafka2.0.0+188	Release notes	Cloudera Distribution of Apache Kafka 2.0.0 Parcel Repository

Cloudera Distribution of Apache Kafka Version	Component	Version	Release Notes	Parcel Repository
1.4.0	Apache Kafka	0.8.2.0+kafka1.4.0+127	Release notes	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	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	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	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 Apache Kafka 1.2.0 Parcel Repository

Table 2: Compatible Release Versions for Cloudera Distribution of Apache Kafka 2.1.0

Type	Location	Parcel File
yum RHEL/CentOS/Oracle 7	http://archive.cloudera.com/kafka/redhat/7/x86_64/kafka/	http://archive.cloudera.com/kafka/parcels/2.1.0/KAFKA-2.1.0-1.2.1.0.p0.115-el7.parcel
yum RHEL/CentOS/Oracle 6	http://archive.cloudera.com/kafka/redhat/6/x86_64/kafka/	http://archive.cloudera.com/kafka/parcels/2.1.0/KAFKA-2.1.0-1.2.1.0.p0.115-el6.parcel
yum RHEL/CentOS/Oracle 5	http://archive.cloudera.com/kafka/redhat/5/x86_64/kafka/	http://archive.cloudera.com/kafka/parcels/2.1.0/KAFKA-2.1.0-1.2.1.0.p0.115-el5.parcel
Debian Jesse 8.2	https://archive.cloudera.com/kafka/debian/jessie/amd64/kafka/dists	http://archive.cloudera.com/kafka/parcels/2.1.0/KAFKA-2.1.0-1.2.1.0.p0.115-jessie.parcel
apt Debian Wheezy 7.0	https://archive.cloudera.com/kafka/debian/wheezy/amd64/kafka/dists	http://archive.cloudera.com/kafka/parcels/2.1.0/KAFKA-2.1.0-1.2.1.0.p0.115-wheezy.parcel
zypper/YaST SLES 12	https://archive.cloudera.com/kafka/sles/12/x86_64/kafka/	http://archive.cloudera.com/kafka/parcels/2.1.0/KAFKA-2.1.0-1.2.1.0.p0.115-sles12.parcel
zypper/YaST SLES 11	https://archive.cloudera.com/kafka/sles/11/x86_64/kafka/	http://archive.cloudera.com/kafka/parcels/2.1.0/KAFKA-2.1.0-1.2.1.0.p0.115-sles11.parcel
Ubuntu Trusty 14.04	https://archive.cloudera.com/kafka/ubuntu/trusty/amd64/kafka/dists	http://archive.cloudera.com/kafka/parcels/2.1.0/KAFKA-2.1.0-1.2.1.0.p0.115-trusty.parcel
Ubuntu Precise 12.04	https://archive.cloudera.com/kafka/ubuntu/precise/amd64/kafka/dists	http://archive.cloudera.com/kafka/parcels/2.1.0/KAFKA-2.1.0-1.2.1.0.p0.115-precise.parcel

Table 3: Compatible Release Versions for Cloudera Distribution of Apache Kafka 1.4.0

Type	Location	Parcel File
yum RHEL/CentOS/Oracle 7	http://archive.cloudera.com/kafka/redhat/7/x86_64/kafka/	http://archive.cloudera.com/kafka/parcels/1.4.0/KAFKA-0.8.2.0-1.kafka1.4.0.p0.56-el5.parcel
yum RHEL/CentOS/Oracle 6	http://archive.cloudera.com/kafka/redhat/6/x86_64/kafka/	http://archive.cloudera.com/kafka/parcels/1.4.0/KAFKA-0.8.2.0-1.kafka1.4.0.p0.56-el6.parcel
yum RHEL/CentOS/Oracle 5	http://archive.cloudera.com/kafka/redhat/5/x86_64/kafka/	http://archive.cloudera.com/kafka/parcels/1.4.0/KAFKA-0.8.2.0-1.kafka1.4.0.p0.56-el7.parcel
apt Debian Wheezy 7.0	https://archive.cloudera.com/kafka/debian/wheezy/amd64/kafka/	http://archive.cloudera.com/kafka/parcels/1.4.0/KAFKA-0.8.2.0-1.kafka1.4.0.p0.56-wheezy.parcel
zypper/YaST SLES	https://archive.cloudera.com/kafka/sles/11/x86_64/kafka/	http://archive.cloudera.com/kafka/parcels/1.4.0/KAFKA-0.8.2.0-1.kafka1.4.0.p0.56-sles11.parcel
Ubuntu Trusty 14.04	https://archive.cloudera.com/kafka/ubuntu/trusty/amd64/kafka/	http://archive.cloudera.com/kafka/parcels/1.4.0/KAFKA-0.8.2.0-1.kafka1.4.0.p0.56-trusty.parcel
Ubuntu Precise 12.04	https://archive.cloudera.com/kafka/ubuntu/precise/amd64/kafka/	http://archive.cloudera.com/kafka/parcels/1.4.0/KAFKA-0.8.2.0-1.kafka1.4.0.p0.56-precise.parcel

Maven Artifacts for Kafka

The following tables lists the project name, groupId, artifactId, and version required to access each Kafka artifact from a Maven POM. For information on how to use Kafka Maven artifacts, see Using the CDH 5 Maven Repository.

The following table lists the project name, groupId, artifactId, and version required to access each Kafka 2.2.0 artifact.

Project	groupId	artifactId	version
Kafka	org.apache.kafka	connect	0.10.2-kafka-2.2.0
	org.apache.kafka	connect-api	0.10.2-kafka-2.2.0
	org.apache.kafka	connect-file	0.10.2-kafka-2.2.0
	org.apache.kafka	connect-json	0.10.2-kafka-2.2.0
	org.apache.kafka	connect-runtime	0.10.2-kafka-2.2.0
	org.apache.kafka	connect-transforms	0.10.2-kafka-2.2.0
	org.apache.kafka	kafka-clients	0.10.2-kafka-2.2.0
	org.apache.kafka	kafka-examples	0.10.2-kafka-2.2.0
	org.apache.kafka	kafka-log4j-appender	0.10.2-kafka-2.2.0
	org.apache.kafka	kafka-streams	0.10.2-kafka-2.2.0
	org.apache.kafka	kafka-streams-examples	0.10.2-kafka-2.2.0
	org.apache.kafka	kafka-tools	0.10.2-kafka-2.2.0
	org.apache.kafka	kafka_2.10	0.10.2-kafka-2.2.0
	org.apache.kafka	kafka_2.11	0.10.2-kafka-2.2.0

The following table lists the project name, groupId, artifactId, and version required to access each Kafka 2.1.1. artifact.

Project	groupId	artifactId	version
Kafka	org.apache.kafka	connect	0.10.0-kafka-2.1.1
Kafka	org.apache.kafka	connect-api	0.10.0-kafka-2.1.1
Kafka	org.apache.kafka	connect-file	0.10.0-kafka-2.1.1
Kafka	org.apache.kafka	connect-json	0.10.0-kafka-2.1.1
Kafka	org.apache.kafka	connect-runtime	0.10.0-kafka-2.1.1
Kafka	org.apache.kafka	kafka-clients	0.10.0-kafka-2.1.1
Kafka	org.apache.kafka	kafka-examples	0.10.0-kafka-2.1.1
Kafka	org.apache.kafka	kafka-log4j-appender	0.10.0-kafka-2.1.1
Kafka	org.apache.kafka	kafka-streams	0.10.0-kafka-2.1.1
Kafka	org.apache.kafka	kafka-streams-examples	0.10.0-kafka-2.1.1
Kafka	org.apache.kafka	kafka-tools	0.10.0-kafka-2.1.1
Kafka	org.apache.kafka	kafka_2.10	0.10.0-kafka-2.1.1
Kafka	org.apache.kafka	kafka_2.11	0.10.0-kafka-2.1.1

The following table lists the project name, groupId, artifactId, and version required to access each Kafka 2.1.0 artifact.

Project	groupId	artifactId	version
Kafka	org.apache.kafka	connect	0.10.0-kafka-2.1.0
Kafka	org.apache.kafka	connect-api	0.10.0-kafka-2.1.0
Kafka	org.apache.kafka	connect-file	0.10.0-kafka-2.1.0
Kafka	org.apache.kafka	connect-json	0.10.0-kafka-2.1.0
Kafka	org.apache.kafka	connect-runtime	0.10.0-kafka-2.1.0
Kafka	org.apache.kafka	kafka-clients	0.10.0-kafka-2.1.0
Kafka	org.apache.kafka	kafka-examples	0.10.0-kafka-2.1.0
Kafka	org.apache.kafka	kafka-log4j-appender	0.10.0-kafka-2.1.0
Kafka	org.apache.kafka	kafka-tools	0.10.0-kafka-2.1.0
Kafka	org.apache.kafka	kafka_2.10	0.10.0-kafka-2.1.0
Kafka	org.apache.kafka	kafka_2.11	0.10.0-kafka-2.1.0

The following table lists the project name, groupId, artifactId, and version required to access each Kafka 2.0.0 artifact.

Project	groupId	artifactId	version
Kafka	org.apache.kafka	connect	0.9.0-kafka-2.0.0
	org.apache.kafka	connect-api	0.9.0-kafka-2.0.0
	org.apache.kafka	connect-file	0.9.0-kafka-2.0.0
	org.apache.kafka	connect-json	0.9.0-kafka-2.0.0
	org.apache.kafka	connect-runtime	0.9.0-kafka-2.0.0
	org.apache.kafka	kafka-clients	0.9.0-kafka-2.0.0
	org.apache.kafka	kafka-examples	0.9.0-kafka-2.0.0
	org.apache.kafka	kafka-log4j-appender	0.9.0-kafka-2.0.0
	org.apache.kafka	kafka-tools	0.9.0-kafka-2.0.0
	org.apache.kafka	kafka_2.10	0.9.0-kafka-2.0.0
	org.apache.kafka	kafka_2.11	0.9.0-kafka-2.0.0

The following table lists the project name, groupId, artifactId, and version required to access each Cloudera Distribution of Apache Kafka 1.4.0 artifact.

Project	groupId	artifactId	version
Kafka	org.apache.kafka	kafka-clients	0.8.2.0-kafka-1.4.0
Kafka	org.apache.kafka	kafka-examples	0.8.2.0-kafka-1.4.0
Kafka	org.apache.kafka	kafka_2.10	0.8.2.0-kafka-1.4.0

Installing, Migrating and Upgrading Kafka

Required Role: Cluster Administrator Full Administrator

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. Before you install, review the Release Notes, particularly:

- What's New in Cloudera Distribution of Apache Kafka? on page 10
- Cloudera Distribution of Apache Kafka Requirements and Supported Versions on page 12
- Known Issues in Cloudera Distribution of Apache Kafka on page 18

Also review the Kafka Product Compatibility Matrix. See http://www.cloudera.com/documentation/enterprise/latest/topics/pcm_kafka.html

Installing or Upgrading Kafka

Required Role Cluster Administrator Full Administrator

Kafka is distributed as a parcel, separate from the CDH parcel. It is also distributed as a package. The steps to install Kafka vary, depending on whether you choose to install from a parcel or a package.

General Information Regarding Installation and Upgrade

Cloudera Manager 5.4 or higher includes the Kafka service. To install, download Kafka using Cloudera Manager, 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 21

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

Note: Upgrade instructions assume you want to upgrade parcel-based Kafka with parcels or package-based Kafka with packages. If you want to switch to using parcel-based Kafka using a Kafka package, you first must uninstall parcel-based Kafka. See Uninstalling an Add-on Service.

Graceful Shutdown of Kafka Brokers

If the Kafka brokers do not shut down gracefully, subsequent restarts may take longer than expected. This can happen when the brokers take longer than 30 seconds to clear their backlog while stopping the Kafka service, stopping the Kafka Broker role, or stopping a cluster where the Kafka service is running. The Kafka brokers are also shut down as part of performing an upgrade. There are two configuration properties you can set to control whether Cloudera Manager waits for the brokers to shut down gracefully:

Table 4: Kafka Shutdown Properties

Property	Description	Default Value
Enable Controlled Shutdown	Enables controlled shutdown of the broker. If enabled, the broker moves all leaders on it to other brokers before shutting itself down. This reduces the unavailability window during shutdown.	Enabled

Property	Description	Default Value
Graceful Shutdown Timeout	The timeout in milliseconds to wait for graceful shutdown to complete.	30000 milliseconds (30 seconds)

To configure these properties, go to **Clusters** > ***Kafka Service*** > **Configuration** and search for "shutdown".

If Kafka is taking a long time for controlled shutdown to complete, consider increasing the value of **Graceful Shutdown Timeout**. Once this timeout is reached, Cloudera Manager issues a forced shutdown, which interrupts the controlled shutdown and could cause subsequent restarts to take longer than expected.

Disks and Filesystem

Cloudera recommends that you use multiple drives to get good throughput. To ensure good latency, do not share the same drives used for Kafka data with application logs or other OS filesystem activity. You can either use RAID to combine these drives into a single volume, or format and mount each drive as its own directory. Since Kafka has replication, RAID can also provide redundancy at the application level. This choice has several tradeoffs.

If you configure multiple data directories, partitions are assigned round-robin to data directories. Each partition is stored entirely in one of the data directories. This can lead to load imbalance between disks if data is not well balanced among partitions.

RAID can potentially do a better job of balancing load between disks because it balances load at a lower level. The primary downside of RAID is that it is usually a big performance hit for write throughput, and it reduces the available disk space.

Another potential benefit of RAID is the ability to tolerate disk failures. However, rebuilding the RAID array is so I/O intensive that it can effectively disable the server, so this does not provide much improvement in availability.

The following table summarizes these pros and cons for RAID10 versus JBOD.

RAID10	JBOD
Can survive single disk failure	Single disk failure kills the broker
Single log directory	More available disk space
Lower total I/O	Higher write throughput
	Broker is not smart about balancing partitions across disk.

Installing or Upgrading Kafka from a Parcel

Required Role Cluster Administrator Full Administrator

1. In Cloudera Manager, select **Hosts** > **Parcels**.
2. If you do not see Kafka in the list of parcels, you can add the parcel to the list.
   a. Find the parcel for the version of Kafka you want to use on Cloudera Distribution of Apache Kafka Versions on page 21.
   b. Copy the parcel repository link.
   c. On the Cloudera Manager **Parcels** page, click **Configuration**.
   d. In the field **Remote Parcel Repository URLs**, click + next to an existing parcel URL to add a new field.
   e. Paste the parcel repository link.
   f. Save your changes.
3. On the Cloudera Manager **Parcels** page, 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.

4. Add the Kafka service to your cluster. See Adding a Service.

Installing or Upgrading Kafka from a Package

Required Role Cluster Administrator Full Administrator

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 21.
3. Install Kafka using the appropriate commands for your operating system.

Table 5: Kafka Installation Commands

Operating System	Commands

RHEL-compatible	`$ sudo yum clean all`
	`$ sudo yum install kafka`
	`$ sudo yum install kafka-server`

SLES	`$ sudo zypper clean --all`
	`$ sudo zypper install kafka`
	`$ sudo zypper install kafka-server`

Ubuntu or Debian	`$ sudo apt-get update`
	`$ sudo apt-get install kafka`
	`$ sudo apt-get install kafka-server`

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 correctly registered to the same ZooKeeper, connect to ZooKeeper using `zookeeper-client`.

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

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

To discover to 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.

Special Considerations When Upgrading from Kafka 1.x to Kafka 2.x

If you upgrade to Kafka 2.0, Cloudera recommends taking the cluster offline because it is a major upgrade with incompatible protocol changes. The upgrade steps are the same even if a cluster is offline.

If taking the cluster offline is not an option, use the following steps to perform a rolling upgrade:

1. In Cloudera Manager, go to the Kafka Configuration page and add `inter.broker.protocol.version=0.8.2.X` to the **Kafka Advanced Configuration Snippet (Safety Valve)**. See Custom Configuration.
2. Upgrade your parcel or package as described in the steps above.
3. Perform a rolling restart.
4. After the entire is upgraded and restarted, remove the property you added in step 1.
5. To have the new protocol take effect, perform another rolling restart.

Upgrade Considerations

- Always upgrade your Kafka cluster before upgrading your clients.
- If using MirrorMaker, upgrade your downstream Kafka clusters first. Otherwise, incompatible messages might be sent downstream.

Special Considerations When Upgrading to Kafka 2.1.x

You must upgrade your Kafka 2.0.x brokers to Kafka 2.1.x before you upgrade your Kafka 2.0.x clients to Kafka 2.1.x.

Migrating from Apache Kafka to Cloudera Distribution of Apache Kafka

Required Role Cluster Administrator Full Administrator

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

Assumptions

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

Table 6: Compatible Release Versions

From Apache Kafka	To Cloudera Distribution of Apache Kafka
0.8.x	1.x
0.9.x	2.x

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 or higher.

Steps for Migrating from Apache Kafka to Cloudera Distribution of Apache Kafka

Cloudera recommends the following migration procedure. You must migrate brokers first, and then 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. Skip the install page by clicking the **Cloudera Manager** icon in the top navigation bar.

Step 1. Migrating Zookeeper

Kafka stores its metadata in ZooKeeper. When 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 will 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 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

All producers, consumers, and Kafka brokers should still be shut down.

1. Back up your `log.dirs` from the old broker machines by copying them to another location or machine.
2. Add the Kafka service to the cluster where you migrated ZooKeeper. See Adding a Service.
3. Add the `broker` role to all machines that were running brokers.
4. Make sure the `kafka` user owns the `log.dirs` files. For example:

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

5. 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. 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` match your old configuration. All of your data and state information is stored here.
6. Start the Kafka brokers using Cloudera Manager.

Step 3. Migrating MirrorMaker

These are the steps for migrating the MirrorMaker role. To avoid compatibility issues, migrate downstream clusters first.

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 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 23.

Using Kafka

This section describes ways you can use Kafka tools to capture data 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 --zookeeper zk01.example.com:2181 --list
sink1
t1
t2
$ /usr/bin/kafka-topics --create --zookeeper hostname:2181/kafka --replication-factor
2
  --partitions 4 --topic topicname
```

- `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` (deprecated)

  > **Note:** kafka-consumer-offset-checker is not supported in the new Consumer API. Use the `ConsumerGroupCommand` tool, below.

  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
```

- `kafka-consumer-groups`

  To view offsets as in the previous example with the `ConsumerOffsetChecker`, you `describe` the consumer group using the following command:

```
$ /usr/bin/kafka-consumer-groups --zookeeper zk01.example.com:2181 --describe --group
flume

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

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.

In CDH 5.7 and higher, the Spark connector to Kafka only works with Kafka 2.0 and higher.

Validating Kafka Integration with Spark Streaming

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

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

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

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

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

Replace the variables as follows:

- *<zkQuorum>* - ZooKeeper quorum URI used by Kafka (for example, `zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181`).
- *<group>* - Consumer group used by the application.
- *<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.

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.

In CDH 5.7 and higher, the Flume connector to Kafka only works with Kafka 2.0 and higher.

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.

For information on configuring Kafka to securely communicate with Flume, see Configuring Flume Security with Kafka.

This topic describes how to configure Kafka sources, sinks, and channels:

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 7: Kafka Source Properties

Property Name	Default Value	Description
type		Must be set to `org.apache.flume.source.kafka.KafkaSource`.
zookeeperConnect		The URI of the ZooKeeper server or quorum used by Kafka. This can be a single host (for example, `zk01.example.com:2181`) or a comma-separated list of hosts in a ZooKeeper quorum (for example, `zk01.example.com:2181,zk02.example.com:2181,zk03.example.com:2181`).
topic		The Kafka topic from which this source reads messages. Flume supports only one topic per source.
groupID	flume	The unique identifier of the Kafka consumer group. Set the same `groupID` in all sources to indicate that they belong to the same consumer group.
batchSize	1000	The maximum number of messages that can be written to a channel in a single batch.
batchDurationMillis	1000	The maximum time (in ms) before a batch is written to the channel. The batch is written when the `batchSize` limit or `batchDurationMillis` limit is reached, whichever comes first.
Other properties supported by the Kafka consumer		Used to configure the Kafka consumer used by the Kafka source. You can use any consumer properties supported by Kafka. Prepend the consumer property name with the prefix `kafka.` (for example, `kafka.fetch.min.bytes`). See the 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 8: 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.
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

Property Name	Default Value	Description
		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.zookeeperConnect = zk01.example.com:2181
tier1.channels.channel1.parseAsFlumeEvent = false
tier1.channels.channel1.topic = channel2
tier1.channels.channel1.consumer.group.id = channel2-grp
tier1.channels.channel1.auto.offset.reset = earliest
tier1.channels.channel1.kafka.bootstrap.servers =
kafka02.example.com:9092,kafka03.example.com:9092
tier1.channels.channel1.transactionCapacity = 1000
tier1.channels.channel1.kafka.consumer.max.partition.fetch.bytes=2097152

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

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

Table 9: 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.

Property Name	Default Value	Description
zookeeperConnect		The URI of the ZooKeeper server or quorum used by Kafka. This can be a single host (for example, `zk01.example.com:2181`) or a comma-separated list of hosts in a ZooKeeper quorum (for example, `zk01.example.com:2181,zk02.example.com:2181, zk03.example.com:2181`).
topic	flume-channel	The Kafka topic the channel will use.
groupID	flume	The unique identifier of the Kafka consumer group the channel uses to register with Kafka.
parseAsFlumeEvent	true	Set to `true` if a Flume source is writing to the channel and expects AvroDataums with the FlumeEvent schema (`org.apache.flume.source.avro.AvroFlumeEvent`) in the channel. Set to `false` if other producers are writing to the topic that the channel is using.
auto.offset.reset	latest	What to do when there is no initial offset in Kafka or if the current offset does not exist on the server (for example, because the data is deleted). • earliest: automatically reset the offset to the earliest offset • latest: automatically reset the offset to the latest offset • none: throw exception to the consumer if no previous offset is found for the consumer's group • anything else: throw exception to the consumer.
kafka.consumer.timeout.ms	100	Polling interval when writing to the sink.
consumer.max.partition.fetch.bytes	1048576	The maximum amount of data per-partition the server will return.
Other properties supported by the Kafka producer		Used to configure the Kafka producer. You can use any producer properties supported by Kafka. Prepend the producer property name with the prefix `kafka.` (for example, `kafka.compression.codec`). See the Kafka documentation for the full list of Kafka producer properties.

Additional Considerations When Using Kafka

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/`.
- Ensure that the JAVA_HOME environment variable is set to your JDK installation directory before using the command-line tools. For example:

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

See the Apache Kafka documentation.

See the Apache 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.

Configuring Kafka Security

This topic describes additional steps you can take to ensure the safety and integrity of your data stored in Kafka, with features available in Cloudera Distribution of Apache Kafka 2.0.0 and higher:

Deploying SSL for Kafka

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

Step 1. Generating Keys and Certificates for Kafka Brokers

First, generate the key and the certificate for each machine in the cluster using the Java keytool utility. See Creating Certificates.

keystore is the keystore file that stores your certificate. *validity* is the valid time of the certificate in days.

```
$ keytool -keystore {tmp.server.keystore.jks} -alias localhost -validity {validity}
-genkey
```

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

Step 2. Creating Your Own Certificate Authority

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

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

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

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

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

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

> **Note:** If you configure Kafka brokers to require client authentication by setting ssl.client.auth to be requested or required on the Kafka brokers config, you must provide a truststore for the Kafka brokers as well. The truststore should have all the CA certificates by which the clients keys are signed.

The keystore created in step 1 stores each machine's own identity. In contrast, the truststore of a client stores all the certificates that the client should trust. Importing a certificate into a truststore means trusting all certificates that are signed by that certificate. This attribute is called the *chain of trust*. It is particularly useful when deploying SSL on a large Kafka cluster. You can sign all certificates in the cluster with a single CA, and have all machines share the same truststore that trusts the CA. That way, all machines can authenticate all other machines.

Step 3. Signing the certificate

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

1. Export the certificate from the keystore:

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

2. Sign it with the CA:

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

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

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

The definitions of the variables are as follows:

- *keystore*: the location of the keystore
- *ca-cert*: the certificate of the CA
- *ca-key*: the private key of the CA
- *ca-password*: the passphrase of the CA
- *cert-file*: the exported, unsigned certificate of the server
- *cert-signed*: the signed certificate of the server

The following Bash script demonstrates the steps described above. One of the commands assumes a password of `test1234`, so either use that password or edit the command before running it.

```
#!/bin/bash
#Step 1
keytool -keystore server.keystore.jks -alias localhost -validity 365 -genkey
#Step 2
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
keytool -keystore server.truststore.jks -alias CARoot -import -file ca-cert
keytool -keystore client.truststore.jks -alias CARoot -import -file ca-cert
#Step 3
keytool -keystore server.keystore.jks -alias localhost -certreq -file cert-file
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days 365
-CAcreateserial -passin pass:test1234
keytool -keystore server.keystore.jks -alias CARoot -import -file ca-cert
keytool -keystore server.keystore.jks -alias localhost -import -file cert-signed
```

Step 4. Configuring Kafka Brokers

Kafka Brokers support listening for connections on multiple ports. You must configure the `listeners` property in `server.properties`, with one or more comma-separated values. If SSL is not enabled for inter-broker communication (see below for how to enable it), both PLAINTEXT and SSL ports are required. For example:

```
listeners=PLAINTEXT://host.name:port,SSL://host.name:port
```

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

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

The following SSL configurations are required on each broker. Each of these values can be set in Cloudera Manager. See Modifying Configuration Properties Using Cloudera Manager:

```
ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
ssl.keystore.password.generator=test1234
ssl.key.password=test1234
ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
ssl.truststore.password.generator=test1234
```

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

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

To enable SSL for inter-broker communication, add the following line to the broker properties file. The default value is `PLAINTEXT`. See Using Kafka Supported Protocols on page 40.

```
security.inter.broker.protocol=SSL
```

Due to import regulations in some countries, the Oracle implementation limits the strength of cryptographic algorithms available by default. If you need stronger algorithms (for example, AES with 256-bit keys), you must obtain the JCE Unlimited Strength Jurisdiction Policy Files and install them in the JDK/JRE. For more information, see the JCA Providers Documentation.

Once you start the broker, you should see the following message in the server.log:

```
with addresses: PLAINTEXT -> EndPoint(192.168.64.1,9092,PLAINTEXT),SSL ->
EndPoint(192.168.64.1,9093,SSL)
```

To check whether the server keystore and truststore are set up properly, run the following command:

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

Note: TLSv1 should be listed under ssl.enabled.protocols.

In the output of this command, you should see the server certificate:

```
-----BEGIN CERTIFICATE-----
{variable sized random bytes}
-----END CERTIFICATE-----
subject=/C=US/ST=CA/L=Santa Clara/O=org/OU=org/CN=John Smith
issuer=/C=US/ST=CA/L=Santa Clara/O=org/OU=org/CN=kafka/emailAddress=test@test.com
```

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

Step 5. Configuring Kafka Clients

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

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

```
security.protocol=SSL
ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks
ssl.truststore.password=test1234
```

If client authentication is required, a keystore must be created as in step 1, and you must also configure the following properties:

```
ssl.keystore.location=/var/private/ssl/kafka.client.keystore.jks
ssl.keystore.password=test1234
ssl.key.password=test1234
```

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

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

Using Kafka Supported Protocols

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

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

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

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

Kafka 2.0 supports the following combinations of protocols.

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

These protocols can be defined for broker-to-client interaction and for broker-to-broker interaction. `security.inter.broker.protocol` allows the broker-to-broker communication protocol to be different than the broker-to-client protocol. It was added to ease the upgrade from non-secure to secure clusters while allowing rolling upgrades.

In most cases, set `security.inter.broker.protocol` to the protocol you are using for broker-to-client communication. Set `security.inter.broker.protocol` to a protocol different than the broker-to-client protocol only when you are performing a rolling upgrade from a non-secure to a secure Kafka cluster.

Enabling Kerberos Authentication

Kafka 2.0 supports Kerberos authentication. If you already have a Kerberos server, you can add Kafka to your current configuration. If you do not have a Kerberos server, install it before proceeding. See Enabling Kerberos Authentication Using the Wizard.

To enable Kerberos authentication for Kafka:

1. Install the CDH Kafka v2.0.x parcel and Cloudera Manager 5.5.3 or higher.
2. From Cloudera Manager, navigate to **Kafka > Configurations**. Set SSL client authentication to none**.** Set **Inter Broker Protocol** to *SASL_PLAINTEXT*.
3. Click **Save Changes**.
4. Restart the Kafka service.
5. Make sure that `listeners = SASL_PLAINTEXT` is present in the Kafka broker logs `/var/log/kafka/server.log`.
6. Create a `jaas.conf` file with the following contents to use with cached Kerberos credentials (you can modify this to use keytab files instead of cached credentials. To generate keytabs, see Step 6: Get or Create a Kerberos Principal for Each User Account).

   If you use kinit first, use this configuration.

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

   If you use keytab, use this configuration:

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
keyTab="/etc/security/keytabs/kafka_server.keytab"
principal="kafka/kafka1.hostname.com@EXAMPLE.COM";
};
```

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

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

8. Test with the Kafka console producer and consumer. To obtain a Kerberos ticket-granting ticket (TGT):

```
$ kinit <user>
```

9. Verify that your topic exists. (This does not use security features, but it is a best practice.)

```
$ kafka-topics --list --zookeeper <zkhost>:2181
```

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

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

11. Run a Kafka console producer.

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

12. Run a Kafka console consumer.

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

Enabling Encryption at Rest

Data encryption is increasingly recognized as an optimal method for protecting data at rest.

Follow these are the steps to encrypt Kafka data that is not in active use.

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

Using Kafka with Sentry Authorization

Starting with Kafka 2.1.x, Apache Sentry includes Kafka binding you can use to enable authorization in Kafka with Sentry. For more information, see Authorization With Apache Sentry.

Configuring Kafka to Use Sentry Authorization

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

Sentry requires that your cluster include HDFS. After you install and start Sentry with the correct configuration, you can stop the HDFS service.

> **Note:** Cloudera's distribution of Kafka does not support LDAP, either through direct LDAP authentication, or via Sentry (when `hadoop.group.mapping` is set to `LdapGroupMapping`).

For more information, see Installing or Upgrading Kafka on page 25 and Sentry Installation.

To configure Sentry authentication for Kafka:

1. Go to **Kafka** > **Configuration**.
2. Select the checkbox **Enable Kerberos Authentication**.
3. Select a Sentry service in the Kafka service configuration.
4. Add **Super users**. Super users can perform any action on any resource in the Kafka cluster. The `kafka` user is added as a super user by default. Super user requests are authorized without going through Sentry, which provides enhanced performance.
5. Select the checkbox **Enable Sentry Privileges Caching** to enhance performance.

Authorizable Resources

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

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

- **Consumergroup**, which controls who can perform consumergroup-level operations such as joining or describing a consumergroup. Its value must exactly match the group.id of a consumergroup.
- **Host**, which controls from where specific operations can be performed. Think of this as a way to achieve IP filtering in Kafka. You can set the value of this resource to the wildcard (*), which represents all hosts.

Authorized Actions

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

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

Authorizing Privileges

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

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

For example, the following are valid privilege strings:

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

Granting Privileges to a Role

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

If Sentry privileges caching is enabled, as recommended, the new privileges you assign take some time to appear in the system. The `time` is the time-to-live interval of the Sentry privileges cache, which is set using `sentry.kafka.caching.ttl.ms`. By default, this interval is 30 seconds. For test clusters, use a lower value, such as 1 ms.

- Allow users in `test-group` to create a new topic from localhost.

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

- Allow users in `test-group` to describe `testTopic` from localhost, which the user creates and uses.

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

- Allow users in `test-group` to write to `testTopic` from localhost, which allows users to produce to `testTopic`.

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

- Create testTopic.

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

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

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

- Allow users in test-group to describe a consumer group, testconsumergroup, that it will start and join.

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

- Allow users in test-group to read from a consumer group, testconsumergroup, that it will start and join.

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

- Allow users in test-group to read from testTopic from localhost and to consume from testTopic.

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

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

```
$ kafka-console-consumer --zookeeper localhost:2181 --topic testTopic --from-beginning
 --consumer.config consumer.properties
This is a message
This is another message
```

Troubleshooting

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

- Make sure you are kinit'd as a user who has privileges to perform an operation.
- Identify which broker is hosting leader of the partition you are trying to produce to or consume from, as this leader is going to authorize your request against Sentry. One easy way of debugging is to just have one Kafka broker. Change log level of the Kafka broker to debug and restart the broker.
- Run the Kafka client or Kafka CLI with required arguments and capture the Kafka log, which should be something like /var/log/kafka/kafka-broker-<HOST_ID>.log on kafka broker's host.
- There will be many Jetty logs, and filtering that out usually helps in reducing noise. Look for log messages from org.apache.sentry.
- Look for following information in the filtered logs:
  - Groups the user Kafka client or CLI is running as.
  - Required privileges for the operation.
  - Retrieved privileges from Sentry.
  - Required and retrieved privileges comparison result.

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

Configuring High Availability and Consistency for Kafka

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 `acks` property on the Producer to `all`. See Acknowledgements on page 45.

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 `acks` property (see Table 8: Kafka Sink Properties on page 34 for details).

Set `acks` to `0` (immediately acknowledge the message without waiting for any brokers to commit), `1` (acknowledge after the leader commits the message), or `all` (acknowledge after all in-sync replicas are committed) according to your requirements. Setting `acks` to `all` 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 `acks` is set to `all`, 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** > **Kakfa 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. The **Topic blacklist** setting has been removed in Cloudera Distribution of Kafka 2.0 and higher.

Note: The **Avoid Data Loss** option from earlier releases has been removed in favor of automatically setting the following properties. Also note that MirrorMaker starts correctly if you enter the numeric values in the configuration snippet (rather than using "max integer" for `retries` and "max long" for `max.block.ms`).

1. Producer settings
   - `acks=all`
   - `retries=2147483647`
   - `max.block.ms=9223372036854775807`
2. Consumer setting
   - `auto.commit.enable=false`
3. MirrorMaker setting
   - `abort.on.send.failure=true`

Configuring Kafka for Performance and Resource Management

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.

This topic describes options that can improve performance and reliability in your Kafka cluster:

Partitions and Memory Usage

For a quick video introduction to load balancing, see tl;dr: Balancing Apache Kafka Clusters.

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 redistirbution 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 limites 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

For a quick video introduction to tuning Kafka, see tl;dr: Tuning Your Apache Kafka Cluster.

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 linger time.

Batch Size

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

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

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

Linger Time

`linger.ms` sets the maximum time to buffer data in asynchronous mode. For example, 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 `linger.ms` to 5 and send more messages in one batch. This would reduce the number of requests sent, but would add up to 5 milliseconds of latency to records sent, even if the load on the system does not warrant the delay.

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

Tuning 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.

Configuring JMX Ephemeral Ports

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

You can change the number for the first port by adding a command similar to `-Dcom.sun.management.jmxremote.rmi.port=<port number>` to the field **Additional Broker Java Options (broker_java_opts)** in Cloudera Manager. The JMX_PORT configuration maps to `com.sun.management.jmxremote.port` by default.

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

Quotas

For a quick video introduction to quotas, see tl;dr: Quotas.

In Cloudera Distribution of Kafka 2.0 and higher, Kafka can enforce quotas on produce and fetch requests. Producers and consumers can use very high volumes of data. This can monopolize broker resources, cause network saturation, and generally deny service to other clients and the brokers themselves. *Quotas* protect against these issues and are important for large, multi-tenant clusters where a small set of clients using high volumes of data can degrade the user experience.

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

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

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

Setting Quotas

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

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

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

To set quotas using Cloudera Manager, open the Kafka **Configuration** page and search for *Quota*. Use the fields provided to set the **Default Consumer Quota** or **Default Producer Quota**. For more information, see Modifying Configuration Properties Using Cloudera Manager.

Setting User Limits for Kafka

Kafka opens many files at the same time. The default setting of 1024 for the maximum number of open files on most Unix-like systems is insufficient. Any significant load can result in failures and cause error messages such as `java.io.IOException...(Too many open files)` to be logged in the Kafka or HDFS log files. You might also notice errors such as this:

```
ERROR Error in acceptor (kafka.network.Acceptor)
java.io.IOException: Too many open files
```

Cloudera recommends setting the value to a relatively high starting point, such as 32,768.

You can monitor the number of file descriptors in use on the Kafka Broker dashboard. In Cloudera Manager:

1. Go to the Kafka service.
2. Select a Kafka Broker.
3. Open **Charts Library** > **Process Resources** and scroll down to the **File Descriptors** chart.

See http://www.cloudera.com/documentation/enterprise/latest/topics/cm_dg_view_charts.html.

Reviewing and Responding to Changes in Kafka 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 *Create Topic* 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.

Metric	Description	Significance	Action
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.
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 ❗ Alert 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 perentage of time this entity has spent in various	Checks the amount of swap memory in use by the role.	Adjust **Process Swap Memory Thresholds**

Metric	Description	Significance	Action
	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.	A failure of this health test might indicate that your machine is overloaded.	monitoring settings for this role instance.
Host Memory Usage	Host memory usage, broken into various usage categories, including *swap.*	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.
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 *-1* to specify there is no limit. Consider adding resources to 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,	If ISR is expanding and shrinking frequently, adjust **Allowed replica lag**.

Metric	Description	Significance	Action
		the expected value for ISR expansion rate is 0.	
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**.

Working with Kafka 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. See Logs.

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.

For example, to restore the default Apache Kafka `log4j` configuration, do the following:

1. Navigate to the Kafka Broker Configuration page.
2. Search for the **Kafka Broker Logging Advanced Configuration Snippet (Safety Valve)** field.
3. Copy and paste the following into the configuration snippet field:

```
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.

For more information on setting properties, see Modifying Configuration Properties Using Cloudera Manager.