Apache Kafka 3

Installing and configuring Apache Kafka

Date of Publish: 2018-08-30

http://docs.hortonworks.com

Contents

Installi	ing Kafka	3
Pre	erequisites	3
Inst	stalling Kafka Using Ambari	.3

Configuring Kafka for a Production Environment	
Preparing the Environment	8
Operating System Settings	
File System Selection	
Disk Drive Considerations	
Java Version	9
Ethernet Bandwidth	9
Customizing Kafka Settings on an Ambari-Managed Cluster	
Kafka Broker Settings	
Connection Settings	
Topic Settings	
Log Settings	
Compaction Settings	
General Broker Settings	
Kafka Producer Settings	
Important Producer Settings	
Kafka Consumer Settings	
Configuring ZooKeeper for Use with Kafka	
Enabling Audit to HDFS for a Secure Cluster	

Installing Kafka

This chapter describes how to install Kafka on an Ambari-managed cluster.

Prerequisites

Before installing Kafka, ZooKeeper must be installed and running on your cluster.

Note that the following underlying file systems are supported for use with Kafka:

- EXT4: supported and recommended
- EXT3: supported

Caution:

Encrypted file systems such as SafenetFS are not supported for Kafka. Index file corruption can occur.

Installing Kafka Using Ambari

Before you install Kafka using Ambari, refer to *Adding a Service* in the Ambari Operations Guide for background information about how to install Hortonworks Data Platform (HDP) components using Ambari. After Kafka is deployed and running, validate the installation. You can use the command-line interface to create a Kafka topic, send test messages, and consume the messages.

Procedure

- 1. Click the Ambari "Services" tab.
- 2. In the Ambari "Actions" menu, select "Add Service." This starts the Add Service wizard, displaying the Choose Services page. Some of the services are enabled by default.
- 3. Scroll through the alphabetic list of components on the Choose Services page, and select "Kafka".

CLUSTER INSTALL WIZARD	Chasse S
Get Started	Choose 3
Select Version	Choose which servi
Install Options	
Confirm Hosts	Service
Choose Services	HDFS
Assign Masters	YARN + MapRedu
Assign Slaves and Clients	III Ter
Oustomize Services	U M
Review	Hive
Install, Start and Test	II HBase
Summary	
	🗆 Pig
	E Sqoop
	Oozie
	☑ ZooKeeper
	E Falcon
	Storm
	Flume
	E Accumulo
	Ambari Metrics
	🗉 Atlas
	2 Kafka
	E Knox
	Log Search
	SmartSense
	Svark

Choose which services you	want to inst	all on your cluster.
Service	Version	Description
II HDFS	2.7.3	Apache Hadoop Distributed File System
U YARN + MapReduce2	2.7.3	Apache Hadoop NextGen MapReduce (YARN)
🔲 Tez	0.7.0	Tez is the next generation Hadoop Query Processing framework written on top of YARN.
E Hive	1.2.1000	Data warehouse system for ad-hoc queries & analysis of large datasets and table & storage management service
II HBase	1.1.2	A Non-relational distributed database, plus Phoenix, a high performance SQL layer for low latency applications.
🗆 Pig	0.16.0	Scripting platform for analyzing large datasets
Sqoop Sqoop	1.4.6	Tool for transferring bulk data between Apache Hadoop and structured data stores such as relational databases
Oozie	4.2.0	System for workflow coordination and execution of Apache Hadoop jobs. This also includes the installation of the optional Oozie Web Console which relies on and will install the ExLIS Library.
ZooKeeper	3.4.6	Centralized service which provides highly reliable distributed coordination
E Falcon	0.10.0	Data management and processing platform
Storm	1.0.1	Apache Hadoop Stream processing framework
E Flume	1.5.2	A distributed service for collecting, aggregating, and moving large amounts of streaming data into HDFS
Accumulo	1.7.0	Robust, scalable, high performance distributed key/value store.
Ambari Metrics	0.1.0	A system for metrics collection that provides storage and retrieval capability for metrics collected from the cluster
Atlas	0.7.0	Atlas Metadata and Governance platform
≥ Kafka	0.10.0	A high-throughput distributed messaging system
Knox	0.9.0	Provides a single point of authentication and access for Apache Hadoop services in a cluster
Log Search	0.5.0	Log aggregation, analysis, and visualization for Ambari managed services. This service is Tech Preview.
SmartSense	1.3.0.0- 960	SmartSense - Hortonworks SmartSense Tool (HST) helps quickly gather configuration, metrics, logs from common HDP services that aids to quickly troubleshoot support cases and receive cluster-specific recommendations.
Spark	1.6.2	Apache Spark is a fast and general engine for large-scale data processing.
Spark2	2.0.0	Apache Spark 2.0 is a fast and general engine for large-scale data processing. This service is Technical Preview .
Zeppelin Notebook	0.6.0	A web-based notebook that enables interactive data analytics. It enables you to make beautiful data-driven, interactive and collaborative documents with SQL, Scala and more.
Mahout	0.9.0	Project of the Apache Software Foundation to produce free implementations of distributed or otherwise scalable machine learning algorithms focused primarily in the areas of collaborative filtering, clustering and classification
Sider	0.91.0	A framework for deploying, managing and monitoring existing distributed applications on YARN.
- Back		Next

- 4. Click Next to continue.
- 5. On the Assign Masters page, review the node assignments for Kafka nodes.

The following screen shows node assignment for a single-node Kafka cluster:

CLUSTER INSTALL WIZARD Get Started	Assign Masters
Select Version	Assign master components to hosts you want to run them on.
Install Options	
Confirm Hosts	ZooKeeper Server: c6401.ambari.apache.org (2.8 GE C6401.ambari.apache.org (2.8 GB, 1 cores)
Choose Services	Kafka Broker: c6401.ambari.apache.org (2.8 GE C
Assign Masters	
Assign Slaves and Clients	
Customize Services	+- Back Next→
Review	
Install, Start and Test	
Summary	

6. If you want Kafka to run with high availability, you must assign more than one node for Kafka brokers, resulting in Kafka brokers running on multiple nodes.

CLUSTER INSTALL WIZARD Get Started	Assign Masters
Select Version	Assign master components to hosts you want to run them on.
Install Options Confirm Hosts	ZooKeeper Server: c6401.ambari.apache.org (2.8 GE C C 6401.ambari.apache.org (2.8 GB, 1 cores)
Choose Services Assign Masters	ZooKeeper Server: c6402.ambari.apache.org (2.8 GE \$ 200Keeper Server: Katka Broker
Assign Slaves and Clients	ZooKeeper Server: c6403.ambari.apache.org (2.8 GE \$ C
Review	Kafka Broker: c6401.ambari.apache.org (2.8 GE C C c6403.ambari.apache.org (2.8 GB, 1 cores)
Install, Start and Test Summary	- Back Next-+

Click the "+" symbol to add more broker nodes to the cluster:

The following screen shows node assignment for a multi-node Kafka cluster:

elect Version	Assign master components to	hosts you want to run them on.	
stall Options onfirm Hosts	ZooKeeper Server:	o6401.ambari.apache.org (2.8 GE 🕯 💿	c6401.ambari.apache.org (2.8 GB, 1 cores)
hoose Services	ZooKeeper Server:	o6402.ambari.apache.org (2.8 GE 🗘 💿	ZooKeeper Server Kafka Broker
usign Masters usign Slaves and Clients	ZooKeeper Server:	o6403.ambari.apache.org (2.8 GE 🕯 💿	c6402.ambari.apache.org (2.8 GB, 1 cores) ZooKeeper Server Kafka Broker
ustomize Services	Kafka Broker:	o6401.ambari.apache.org (2.8 GE 🕯 😋	o5403.ambari.apache.oro (2.8 GB. 1 cores)
stall, Start and Test	Kafka Broker:	o6402.ambari.apache.org (2.8 GE 🕏 💿	ZooKeeper Server Kafka Broker
immary	Kafka Broker:	o6403.ambari.apache.org (2.8 GE 🕯 💽	

5

- 7. Click **Next** to continue.
- 8. On the Assign Slaves and Clients page, choose the nodes that you want to run ZooKeeper clients:

CLUSTER INSTALL WIZARD	Assign Slaves and Clients	
Select Version Install Options	Assign slave and client components to hosts you want to run them on. Hosts that are assigned master components are shown with . "Client" will install ZooKeeper Client	
Choose Services	Host	all none
Assign Slaves and Clients Customize Services	c6402.ambari.apache.org *	Client
Review Install, Start and Test	c6403.ambari.apache.org *	Client
Summary	- Back	Show: 25 t 1-3of3 H ← → H

9. Click Next to continue.

10. Ambari displays the Customize Services page, which lists a series of services:

CLUSTER INSTALL WIZARD Get Started Select Version	Customize Services	
Install Options Confirm Hosts Choose Services	We have come up with recommended configurations for the services you selected. Customize them as you see fit. ZooKeeper Kafka Misc	
Assign Masters Assign Slaves and Clients Customize Services	Group Default (1) Manage Config Groups Filter Kafka Broker	•
Review Install, Start and Test Summary	Advanced kafka-broker	
	Advanced kafka-env Advanced kafka-log4j	
	Custom kafka-broker	
	ef All configurations have been addressed. ← Back	Next →

For your initial configuration you should use the default values set by Ambari. If Ambari prompts you with the message "Some configurations need your attention before you can proceed," review the list of properties and provide the required information.

11. Click Next to continue.

12. When the wizard displays the Review page, ensure that all HDP components correspond to HDP 2.5 or later:



13. Click **Deploy** to begin installation.

14. Ambari displays the Install, Start and Test page. Monitor the status bar and messages for progress updates:

elect Version Istall Options Confirm Hosts	Please wait while the selected services a	re installed and started.	100 % overall
nstall Options Confirm Hosts			100 % overall
onfirm Hosts			100 % overall
unes Renines		-	_
0000 00 1000		Show: A	(0) In Progress (3) Warning (3) Success (1) Eal.(
sign Masters	Host	Status	Message
sign Slaves and Clients	c6401.ambari.apache.org	100%	Success
stomize Services	1 of 1 hosts showing - Show All		Show: 25 1 1-1011 N 6 9
larw			
all, Start and Test	Successfully installed and started the ser	vices.	
mmary			

15. When the wizard presents a summary of results, click "Complete" to finish installing Kafka:

GLUSTER INSTALL WIZARD Get Started	Summary
Select Version	Here is the summary of the install process.
Install Options	
Confirm Hosts	The cluster consists of 2 hosts
Choose Services	Installed and started services successfully on 2 new hosts Master services installed
Assign Masters	All services started
Assign Slaves and Clients	All tests passed Install and start completed in 696 minutes and 46 seconds
Customize Services	
Review	
Install, Start and Test	Complete
Summary	

What to do next

After Kafka is deployed and running, validate the installation. You can use the command-line interface to create a Kafka topic, send test messages, and consume the messages.

Configuring Kafka for a Production Environment

This chapter covers topics related to Kafka configuration, including:

- Preparing the environment
- Customizing settings for brokers, producers, and consumers
- Configuring ZooKeeper for use with Kafka
- Enabling audit to HDFS when running Kafka on a secure cluster

Preparing the Environment

The following factors can affect Kafka performance:

- Operating system settings
- File system selection
- Disk drive configuration
- Java version
- Ethernet bandwidth

Operating System Settings

Consider the following when configuring Kafka:

- Kafka uses page cache memory as a buffer for active writers and readers, so after you specify JVM size (using Xmx and -Xms Java options), leave the remaining RAM available to the operating system for page caching.
- Kafka needs open file descriptors for files and network connections. You should set the file descriptor limit to at least 128000.
- You can increase the maximum socket buffer size to enable high-performance data transfer.

File System Selection

Kafka uses regular Linux disk files for storage. We recommend using the EXT4 or XFS file system. Improvements to the XFS file system show improved performance characteristics for Kafka workloads without compromising stability.



Caution:

- Do not use mounted shared drives or any network file systems with Kafka, due to the risk of index failures and (in the case of network file systems) issues related to the use of MemoryMapped files to store the offset index.
- Encrypted file systems such as SafenetFS are not supported for Kafka. Index file corruption can occur.

Disk Drive Considerations

For throughput, we recommend dedicating multiple drives to Kafka data. More drives typically perform better with Kafka than fewer. Do not share these Kafka drives with any other application or use them for Kafka application logs.

You can configure multiple drives by specifying a comma-separated list of directories for the log.dirs property in the server.properties file. Kafka uses a round-robin approach to assign partitions to directories specified in log.dirs; the default value is /tmp/kafka-logs.

The num.io.threads property should be set to a value equal to or greater than the number of disks dedicated for Kafka. Recommendation: start by setting this property equal to the number of disks.

Depending on how you configure flush behavior (see "Log Flush Management"), a faster disk drive is beneficial if the log.flush.interval.messages property is set to flush the log file after every 100,000 messages (approximately).

Kafka performs best when data access loads are balanced among partitions, leading to balanced loads across disk drives. In addition, data distribution across disks is important. If one disk becomes full and other disks have available space, this can cause performance issues. To avoid slowdowns or interruptions to Kafka services, you should create usage alerts that notify you when available disk space is low.

RAID can potentially improve load balancing among the disks, but RAID can cause performance bottleneck due to slower writes. In addition, it reduces available disk space. Although RAID can tolerate disk failures, rebuilding RAID array is I/O-intensive and effectively disables the server. Therefore, RAID does not provide substantial improvements in availability.

Java Version

With Apache Kafka on HDP 2.5, you should use the latest update for Java version 1.8 and make sure that G1 garbage collection support is enabled. (G1 support is enabled by default in recent versions of Java.) If you prefer to use Java 1.7, make sure that you use update u51 or later.

Here are several recommended settings for the JVM:

```
-Xmx6g
-Xms6g
-XX:MetaspaceSize=96m
-XX:+UseG1GC
-XX:MaxGCPauseMillis=20
-XX:InitiatingHeapOccupancyPercent=35
-XX:G1HeapRegionSize=16M
-XX:MinMetaspaceFreeRatio=50
-XX:MaxMetaspaceFreeRatio=80
```

To set JVM heap size for the Kafka broker, export KAFKA_HEAP_OPTS; for example:

export KAFKA_HEAP_OPTS="-Xmx2g -Xms2g" ./kafka-server-start.sh

Ethernet Bandwidth

Ethernet bandwidth can have an impact on Kafka performance; make sure it is sufficient for your throughput requirements.

Customizing Kafka Settings on an Ambari-Managed Cluster

To customize configuration settings during the Ambari installation process, click the "Kafka" tab on the Customize Services page:

CLUSTER INSTALL WIZARD	Customize Se	ervices				
Select Version	We have come up with reco	mmended configurations for the services you selected. Custo	mize them as	you s	ee fit	L
Install Options Confirm Hosts	2					
Choose Services	zookeeper kaska Misc					
Assign Masters	Group Default (1) •	Manage Config Groups	Filter			•
Customize Services	 Kafka Broker 					
Neview	Kafka Broker host	o6401.ambari.apache.org				
lummary	zookeeper.connect	c6401.ambari.apache.org:2181			•	e
	log.roll.hours	168			۰	c
	log.retention.hours	168		8	۰	e
	log.dirs	/kafka-logs			•	c
	listeriors	PLAINTEXT://ocalhost:6867			•	c

If you want to access configuration settings after installing Kafka using Ambari:

- 1. Click Kafka on the Ambari dashboard.
- 2. Choose Configs.

To view and modify settings, either scroll through categories and expand a category (such as "Kafka Broker", as shown in the graphic), or use the "Filter" box to search for a property.

Settings in the Advanced kafka-env category are configured by Ambari; you should not modify these settings:

 Advanced kafka-env 				
is_supported_kafka_ ranger	true	0	c	
kafka_keytab		0		
afka_log_dir	/var/log/kafka	۰	с	
Kafka PID dir	/var/run/kafka	c		
kafka_principal_name		۰		
kafka_user_nofile_limit	128000	۰	с	
kafka_user_nproc_limit	65536	0	с	
afka-env template	<pre>#!/bin/bash # Set KAFKA specific environment variables here. # The java implementation to use. export JAVA_HOME={{java64_home}} export PATH=\$PATH:\$JAVA_HOME/bin export PID_DIR={{kafka_pid_dir}} export LOG_DIR={{kafka_pid_dir}} export KAFKA_KERBEROS_PARAMS={{kafka_kerberos_params}} # Add kafka sink to classpath and related depenencies if [-e */usr/lib/ambari-metrics-kafka-sink/ambari-metrics-kafka-sink.jar*]; export CLASSPATH=\$CLASSPATH:/usr/lib/ambari-metrics-kafka-sink/lib fi if [-f /etc/kafka/conf/kafka-ranger-env.sh]; then . /etc/kafka/conf/kafka-ranger-env.sh fi</pre>	then nbari-		o c

To add configuration properties that are not listed by default in Ambari, navigate to the Custom kafka-broker category:

CLUSTER	Add Propert	у	×
Get Start	Туре	kafka-broker.xml	••
Install Op	Key	1	
Confirm H	Value		
Assign M Assign Si			
Customiz Review			Cancel Add
	int and Test	Advanced kafka-broker	
		Advanced kafka-env	
		Advanced kafka-log4j	
		Custom kafka-broker	

Kafka Broker Settings

The following subsections describe configuration settings that influence the performance of Kafka brokers.

Connection Settings

Review the following connection setting in the Advanced kafka-broker category, and modify as needed:

zookeeper.session.timeout.ms	Specifies ZooKeeper session timeout, in milliseconds. The default value is 30000 ms.
	If the server fails to signal heartbeat to ZooKeeper within this period of time, the server is considered to be dead. If you set this value too low, the server might be falsely considered dead; if you set it too high it may take too long to recognize a truly dead server.
	If you see frequent disconnection from the ZooKeeper server, review this setting. If long garbage collection pauses cause Kafka to lose its ZooKeeper session, you might need to configure longer timeout values.
advertised.listeners	If you have manually set listeners to advertised.listeners=PLAINTEXT://\$HOSTNAME: \$PORT, after enabling Kerberos, change the listener configuration to advertised.listeners= SASL_PLAINTEXT://\$HOSTNAME:\$PORT.

Important:

Do not change the following connection settings:

zookeeper.connect

A comma-separated list of ZooKeeper hostname:port pairs. Ambari sets this value. Do not change this setting.

Topic Settings

For each topic, Kafka maintains a structured commit log with one or more partitions. These topic partitions form the basic unit of parallelism in Kafka. In general, the more partitions there are in a Kafka cluster, the more parallel consumers can be added, resulting in higher throughput.

You can calculate the number of partitions based on your throughput requirements. If throughput from a producer to a single partition is P and throughput from a single partition to a consumer is C, and if your target throughput is T, the minimum number of required partitions is

max (T/P, T/C).

Note also that more partitions can increase latency:

- End-to-end latency in Kafka is defined as the difference in time from when a message is published by the producer to when the message is read by the consumer.
- Kafka only exposes a message to a consumer after it has been committed, after the message is replicated to all insync replicas.
- Replication of one thousand partitions from one broker to another can take up 20ms. This is too long for some real-time applications.
- In the new Kafka producer, messages are accumulated on the producer side; producers buffer the message per partition. This approach allows users to set an upper bound on the amount of memory used for buffering incoming messages. After enough data is accumulated or enough time has passed, accumulated messages are removed and sent to the broker. If you define more partitions, messages are accumulated for more partitions on the producer side.
- Similarly, the consumer fetches batches of messages per partition. Consumer memory requirements are proportional to the number of partitions that the consumer subscribes to.

Important Topic Properties

Review the following settings in the Advanced kafka-broker category, and modify as needed:

auto.create.topics.enable	Enable automatic creation of topics on the server. If this property is set to true, then attempts to produce, consume, or fetch metadata for a nonexistent topic automatically create the topic with the default replication factor and number of partitions. The default is enabled.
default.replication.factor	Specifies default replication factors for automatically created topics. For high availability production systems, you should set this value to at least 3.
num.partitions	Specifies the default number of log partitions per topic, for automatically created topics. The default value is 1. Change this setting based on the requirements related to your topic and partition design.
delete.topic.enable	Allows users to delete a topic from Kafka using the admin tool, for Kafka versions 0.9 and later. Deleting a topic through the admin tool will have no effect if this setting is turned off.
	By default this feature is turned off (set to false).

Log Settings

Review the following settings in the Kafka Broker category, and modify as needed:

log.roll.hours	The maximum time, in hours, before a new log segment is rolled out. The default value is 168 hours (seven days).	
	This setting controls the period of time after which Kafka will force the log to roll, even if the segment file is not full. This ensures that the retention process is able to delete or compact old data.	
log.retention.hours	The number of hours to keep a log file before deleting it. The default value is 168 hours (seven days).	
	When setting this value, take into account your disk space and how long you would like messages to be available. An active consumer can read quickly and deliver messages to their destination.	
	The higher the retention setting, the longer the data will be preserved. Higher settings generate larger log files, so increasing this setting might reduce your overall storage capacity.	
log.dirs	A comma-separated list of directories in which log data is kept. If you have multiple disks, list all directories under each disk.	
Review the following setting in the Advanced kafka-broker category, and modify as needed:		
los notorition botos		

log.retention.bytes	The amount of data to retain in the log for each topic partition. By default, log size is unlimited.
	Note that this is the limit for each partition, so multiply this value by the number of partitions to calculate the total data retained for the topic.
	If log.retention.hours and log.retention.bytes are both set, Kafka deletes a segment when either limit is exceeded.
log.segment.bytes	The log for a topic partition is stored as a directory of segment files. This setting controls the maximum size of a segment file before a new segment is rolled over in the log. The default is 1 GB.

Log Flush Management

Kafka writes topic messages to a log file immediately upon receipt, but the data is initially buffered in page cache. A log flush forces Kafka to flush topic messages from page cache, writing the messages to disk.

We recommend using the default flush settings, which rely on background flushes done by Linux and Kafka. Default settings provide high throughput and low latency, and they guarantee recovery through the use of replication.

If you decide to specify your own flush settings, you can force a flush after a period of time, or after a specified number of messages, or both (whichever limit is reached first). You can set property values globally and override them on a per-topic basis.

There are several important considerations related to log file flushing:

• Durability: unflushed data is at greater risk of loss in the event of a crash. A failed broker can recover topic partitions from its replicas, but if a follower does not issue a fetch request or consume from the leader's log-end

offset within the time specified by replica.lag.time.max.ms (which defaults to 10 seconds), the leader removes the follower from the in-sync replica ("ISR"). When this happens there is a slight chance of message loss if you do not explicitly set log.flush.interval.messages. If the leader broker fails and the follower is not caught up with the leader, the follower can still be under ISR for those 10 seconds and messages during leader transition to follower can be lost.

- Increased latency: data is not available to consumers until it is flushed (the fsync implementation in most Linux filesystems blocks writes to the file system).
- Throughput: a flush operation is typically an expensive operation.
- Disk usage patterns are less efficient.
- Page-level locking in background flushing is much more granular.

log.flush.interval.messages specifies the number of messages to accumulate on a log partition before Kafka forces a flush of data to disk.

log.flush.scheduler.interval.ms specifies the amount of time (in milliseconds) after which Kafka checks to see if a log needs to be flushed to disk.

log.segment.bytes specifies the size of the log file. Kafka flushes the log file to disk whenever a log file reaches its maximum size.

log.roll.hours specifies the maximum length of time before a new log segment is rolled out (in hours); this value is secondary to log.roll.ms. Kafka flushes the log file to disk whenever a log file reaches this time limit.

Compaction Settings

Review the following settings in the Advanced kafka-broker category, and modify as needed:

log.cleaner.dedupe.buffer.size	Specifies total memory used for log deduplication across all cleaner threads.
	By default, 128 MB of buffer is allocated. You may want to review this and other log.cleaner configuration values, and adjust settings based on your use of compacted topics (consumer_offsets and other compacted topics).
log.cleaner.io.buffer.size	Specifies the total memory used for log cleaner I/O buffers across all cleaner threads. By default, 512 KB of buffer is allocated. You may want to review this and other log.cleaner configuration values, and adjust settings based on your usage of compacted topics (consumer_offsets and other compacted topics).

General Broker Settings

Review the following settings in the Advanced kafka-broker category, and modify as needed:

auto.leader.rebalance.enable	Enables automatic leader balancing. A background thread checks and triggers leader balancing (if needed) at regular intervals. The default is enabled.
unclean.leader.election.enable	This property allows you to specify a preference of availability or durability. This is an important setting: If availability is more important than avoiding data loss, ensure that this property is set to true. If preventing data loss is more important than availability, set this property to false.
	This setting operates as follows:

	 If unclean.leader.election.enable is set to true (enabled), an out-of-sync replica will be elected as leader when there is no live in-sync replica (ISR). This preserves the availability of the partition, but there is a chance of data loss. If unclean.leader.election.enable is set to false and there are no live in-sync replicas, Kafka returns an error and the partition will be unavailable.
	This property is set to true by default, which favors availability.
	If durability is preferable to availability, set unclean.leader.election to false.
controlled.shutdown.enable	Enables controlled shutdown of the server. The default is enabled.
min.insync.replicas	When a producer sets acks to "all", min.insync.replicas specifies the minimum number of replicas that must acknowledge a write for the write to be considered successful. If this minimum cannot be met, then the producer will raise an exception.
	When used together, min.insync.replicas and producer acks allow you to enforce stronger durability guarantees.
	You should set min.insync.replicas to 2 for replication factor equal to 3.
message.max.bytes	Specifies the maximum size of message that the server can receive. It is important that this property be set with consideration for the maximum fetch size used by your consumers, or a producer could publish messages too large for consumers to consume.
	Note that there are currently two versions of consumer and producer APIs. The value of message.max.bytes must be smaller than the max.partition.fetch.bytes setting in the new consumer, or smaller than the fetch.message.max.bytes setting in the old consumer. In addition, the value must be smaller than replica.fetch.max.bytes.
replica.fetch.max.bytes	Specifies the number of bytes of messages to attempt to fetch. This value must be larger than message.max.bytes.
broker.rack	The rack awareness feature distributes replicas of a partition across different racks. You can specify that a broker belongs to a particular rack through the "Custom kafka-broker" menu option. For more information about the rack awareness feature, see http://kafka.apache.org/documentation.html#basic_ops_racks.

Kafka Producer Settings

If performance is important and you have not yet upgraded to the new Kafka producer (client version 0.9.0.1 or later), consider doing so. The new producer is generally faster and more fully featured than the previous client.

To use the new producer client, add the associated maven dependency on the client jar; for example:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>0.9.0.0</version>
</dependency>
```

For more information, see the KafkaProducer javadoc.

The following subsections describe several types of configuration settings that influence the performance of Kafka producers.

Important Producer Settings

The lifecycle of a request from producer to broker involves several configuration settings:

- 1. The producer polls for a batch of messages from the batch queue, one batch per partition. A batch is ready when one of the following is true:
 - batch.size is reached. Note: Larger batches typically have better compression ratios and higher throughput, but they have higher latency.
 - linger.ms (time-based batching threshold) is reached. Note: There is no simple guideilne for setting linger.ms values; you should test settings on specific use cases. For small events (100 bytes or less), this setting does not appear to have much impact.
 - Another batch to the same broker is ready.
 - The producer calls flush() or close().
- 2. The producer groups the batch based on the leader broker.
- 3. The producer sends the grouped batch to the broker.

The following paragraphs list additional settings related to the request lifecycle:

max.in.flight.requests.per.connection (pipelining)	The maximum number of unacknowledged requests the client will send on a single connection before blocking. If this setting is greater than 1, pipelining is used when the producer sends the grouped batch to the broker. This improves throughput, but if there are failed sends there is a risk of out-of-order delivery due to retries (if retries are enabled). Note also that excessive pipelining reduces throughput.
compression.type	Compression is an important part of a producer's work, and the speed of different compression types differs a lot.
	To specify compression type, use the compression.type property. It accepts standard compression codecs ('gzip', 'snappy', 'lz4'), as well as 'uncompressed' (the default, equivalent to no compression), and 'producer' (uses the compression codec set by the producer).
	Compression is handled by the user thread. If compression is slow it can help to add more threads. In addition, batching efficiency impacts the compression ratio: more batching leads to more efficient compression.

acks

Configuring Kafka for a Production Environment

The acks setting specifies acknowledgments that the producer requires the leader to receive before considering a request complete. This setting defines the durability level for the producer.

Acks	Throughput	Latency	Durability
0	High	Low	No Guarantee. The producer does not wait for acknowledgmen from the server.
1	Medium	Medium	Leader writes the record to its local log, and responds without awaiting full acknowledgment from all followers.
-1	Low	High	Leader waits for the full set of in- sync replicas (ISRs) to acknowledge the record. This guarantees that the record is not lost as long as at least one IRS is active.

The new Producer API supports an optional flush() call, which makes all buffered records immediately available to send (even if linger.ms is greater than 0).

When using flush(), the number of bytes between two flush() calls is an important factor for performance.

- In microbenchmarking tests, a setting of approximately 4MB performed well for events 1KB in size.
- A general guideline is to set batch.size equal to the total bytes between flush()calls divided by number of partitions:

(total bytes between flush()calls) / (partition count)

Additional Considerations

A producer thread going to the same partition is faster than a producer thread that sends messages to multiple partitions.

If a producer reaches maximum throughput but there is spare CPU and network capacity on the server, additional producer processes can increase overall throughput.

Performance is sensitive to event size: larger events are more likely to have better throughput. In microbenchmarking tests, 1KB events streamed faster than 100-byte events.

flush()

Kafka Consumer Settings

You can usually obtain good performance from consumers without tuning configuration settings. In microbenchmarking tests, consumer performance was not as sensitive to event size or batch size as was producer performance. Both 1KG and 100B events showed similar throughput.

One basic guideline for consumer performance is to keep the number of consumer threads equal to the partition count.

Configuring ZooKeeper for Use with Kafka

Here are several recommendations for ZooKeeper configuration with Kafka:

- Do not run ZooKeeper on a server where Kafka is running.
- When using ZooKeeper with Kafka you should dedicate ZooKeeper to Kafka, and not use ZooKeeper for any other components.
- Make sure you allocate sufficient JVM memory. A good starting point is 4GB.
- To monitor the ZooKeeper instance, use JMX metrics.

Configuring ZooKeeper for Multiple Applications

If you plan to use the same ZooKeeper cluster for different applications (such as Kafka cluster1, Kafka cluster2, and HBase), you should add a chroot path so that all Kafka data for a cluster appears under a specific path.

The following example shows a sample chroot path:

c6401.ambari.apache.org:2181:/kafka-root, c6402.ambari.apache.org:2181:/kafka-root

You must create this chroot path yourself before starting the broker, and consumers must use the same connection string.

Enabling Audit to HDFS for a Secure Cluster

To enable audit to HDFS when running Storm on a secure cluster, perform the steps listed at the bottom of *Manually Updating Ambari HDFS Audit Settings* in the HDP Security Guide.