APACHE kafka
A distributed streaming platform

Kafka 2.4 Documentation

1. GETTING STARTED

- 1.1 Introduction
- 1.2 Use Cases
- 1.3 Quick Start
- 1.4 Ecosystem
- 1.5 Upgrading

2. APIS

- 2.1 Producer API
- 2.2 Consumer API
- 2.3 Streams API
- 2.4 Connect API
- 2.5 Admin API

3. CONFIGURATION

- 3.1 Broker Configs
- 3.2 Topic Configs
- 3.3 Producer Configs
- 3.4 Consumer Configs
- 3.5 Kafka Connect Configs
- 3.6 Kafka Streams Configs
- 3.7 AdminClient Configs

4. DESIGN

- 4.1 Motivation
- 4.2 Persistence
- 4.3 Efficiency
- 4.4 The Producer
- 4.5 The Consumer
- 4.6 Message Delivery Semantics

- 4.7 Replication
- 4.8 Log Compaction
- 4.9 Quotas

5. IMPLEMENTATION

- 5.1 Network Layer
- 5.2 Messages
- 5.3 Message format
- 5.4 Log
- 5.5 Distribution

6. OPERATIONS

- 6.1 Basic Kafka Operations
  - Adding and removing topics
  - Modifying topics
  - Graceful shutdown
  - Balancing leadership
  - Checking consumer position
  - Mirroring data between clusters
  - Expanding your cluster
  - Decommissioning brokers
  - Increasing replication factor
- 6.2 Datacenters
- 6.3 Important Configs
  - Important Client Configs
  - A Production Server Configs
- 6.4 Java Version
- 6.5 Hardware and OS
  - OS
  - Disks and Filesystems
  - Application vs OS Flush Management
  - Linux Flush Behavior
  - Ext4 Notes
- 6.6 Monitoring
  - Selector Monitoring
  - Common Node Monitoring
  - Producer Monitoring

    - Consumer Monitoring
    - Connect Monitoring
    - Streams Monitoring
    - Others
  - 6.7 ZooKeeper
    - Stable Version
    - Operationalization

7. SECURITY
  - 7.1 Security Overview
  - 7.2 Encryption and Authentication using SSL
  - 7.3 Authentication using SASL
  - 7.4 Authorization and ACLs
  - 7.5 Incorporating Security Features in a Running Cluster
  - 7.6 ZooKeeper Authentication
    - New Clusters
    - Migrating Clusters
    - Migrating the ZooKeeper Ensemble

8. KAFKA CONNECT
  - 8.1 Overview
  - 8.2 User Guide
    - Running Kafka Connect
    - Configuring Connectors
    - Transformations
    - REST API
  - 8.3 Connector Development Guide

9. KAFKA STREAMS
  - 9.1 Play with a Streams Application
  - 9.2 Write your own Streams Applications
  - 9.3 Developer Manual
  - 9.4 Core Concepts
  - 9.5 Architecture
  - 9.6 Upgrade Guide

1. GETTING STARTED

1.1 Introduction

Apache Kafka® is *a distributed streaming platform*. What exactly does that mean?

A streaming platform has three key capabilities:

- Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system.
- Store streams of records in a fault-tolerant durable way.
- Process streams of records as they occur.

Kafka is generally used for two broad classes of applications:

- Building real-time streaming data pipelines that reliably get data between systems or applications
- Building real-time streaming applications that transform or react to the streams of data

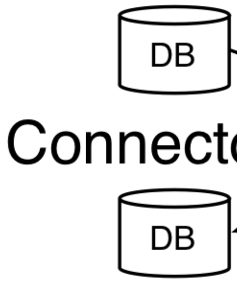
To understand how Kafka does these things, let's dive in and explore Kafka's capabilities from the bottom up.

First a few concepts:

- Kafka is run as a cluster on one or more servers that can span multiple datacenters.
- The Kafka cluster stores streams of *records* in categories called *topics*.
- Each record consists of a key, a value, and a timestamp.

Kafka has four core APIs:

- The Producer API allows an application to publish a stream of records to one or more Kafka topics.
- The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them.
- The Streams API allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.



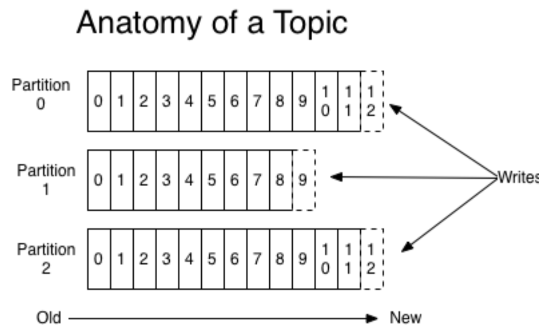
In Kafka the communication between the clients and the servers is done with a simple, high-performance, languag
backwards compatibility with older version. We provide a Java client for Kafka, but clients are available in many la

Topics and Logs

Let's first dive into the core abstraction Kafka provides for a stream of records—the topic.

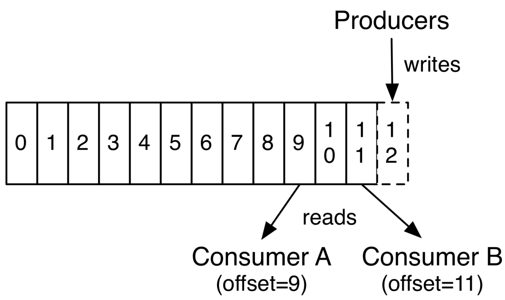
A topic is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber; tha
to the data written to it.

For each topic, the Kafka cluster maintains a partitioned log that looks like this:



Each partition is an ordered, immutable sequence of records that is continually appended to—a structured commit
number called the *offset* that uniquely identifies each record within the partition.

The Kafka cluster durably persists all published records—whether or not they have been consumed—using a config
to two days, then for the two days after a record is published, it is available for consumption, after which it will be d
constant with respect to data size so storing data for a long time is not a problem.



In fact, the only metadata retained on a per-consumer basis is the offset or position of that consumer in the log. Th
advance its offset linearly as it reads records, but, in fact, since the position is controlled by the consumer it can co
reset to an older offset to reprocess data from the past or skip ahead to the most recent record and start consumi

This combination of features means that Kafka consumers are very cheap—they can come and go without much i
use our command line tools to "tail" the contents of any topic without changing what is consumed by any existing

The partitions in the log serve several purposes. First, they allow the log to scale beyond a size that will fit on a sin
it, but a topic may have many partitions so it can handle an arbitrary amount of data. Second they act as the unit o

Distribution

The partitions of the log are distributed over the servers in the Kafka cluster with each server handling data and re
across a configurable number of servers for fault tolerance.

Each partition has one server which acts as the "leader" and zero or more servers which act as "followers". The lea
followers passively replicate the leader. If the leader fails, one of the followers will automatically become the new l
follower for others so load is well balanced within the cluster.

Geo-Replication

Kafka MirrorMaker provides geo-replication support for your clusters. With MirrorMaker, messages are replicated a
active/passive scenarios for backup and recovery; or in active/active scenarios to place data closer to your users,

Producers

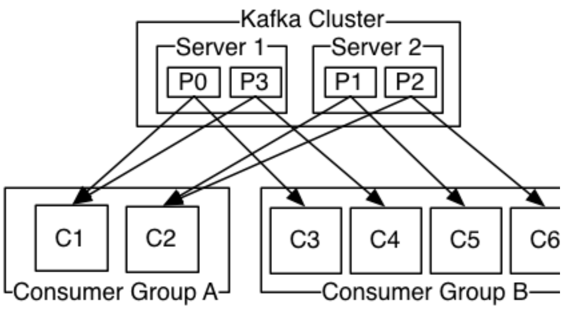
Producers publish data to the topics of their choice. The producer is responsible for choosing which record to ass
robin fashion simply to balance load or it can be done according to some semantic partition function (say based o
second!

Consumers

Consumers label themselves with a *consumer group* name, and each record published to a topic is delivered to on
Consumer instances can be in separate processes or on separate machines.

If all the consumer instances have the same consumer group, then the records will effectively be load balanced ov

If all the consumer instances have different consumer groups, then each record will be broadcast to all the consur



A two server Kafka cluster hosting four partitions (P0-P3) with two consumer groups. Consumer group A has two

More commonly, however, we have found that topics have a small number of consumer groups, one for each "logic
instances for scalability and fault tolerance. This is nothing more than publish-subscribe semantics where the sub

The way consumption is implemented in Kafka is by dividing up the partitions in the log over the consumer instanc
share" of partitions at any point in time. This process of maintaining membership in the group is handled by the Ka
take over some partitions from other members of the group; if an instance dies, its partitions will be distributed to

Kafka only provides a total order over records *within* a partition, not between different partitions in a topic. Per-par sufficient for most applications. However, if you require a total order over records this can be achieved with a topic consumer process per consumer group.

Multi-tenancy

You can deploy Kafka as a multi-tenant solution. Multi-tenancy is enabled by configuring which topics can produce Administrators can define and enforce quotas on requests to control the broker resources that are used by clients.

Guarantees

At a high-level Kafka gives the following guarantees:

- Messages sent by a producer to a particular topic partition will be appended in the order they are sent. That is, i is sent first, then M1 will have a lower offset than M2 and appear earlier in the log.
- A consumer instance sees records in the order they are stored in the log.
- For a topic with replication factor N, we will tolerate up to N-1 server failures without losing any records commit

More details on these guarantees are given in the design section of the documentation.

Kafka as a Messaging System

How does Kafka's notion of streams compare to a traditional enterprise messaging system?

Messaging traditionally has two models: queuing and publish-subscribe. In a queue, a pool of consumers may rea subscribe the record is broadcast to all consumers. Each of these two models has a strength and a weakness. Th processing of data over multiple consumer instances, which lets you scale your processing. Unfortunately, queues Publish-subscribe allows you broadcast data to multiple processes, but has no way of scaling processing since ev

The consumer group concept in Kafka generalizes these two concepts. As with a queue the consumer group allow members of the consumer group). As with publish-subscribe, Kafka allows you to broadcast messages to multiple

The advantage of Kafka's model is that every topic has both these properties—it can scale processing and is also

Kafka has stronger ordering guarantees than a traditional messaging system, too.

A traditional queue retains records in-order on the server, and if multiple consumers consume from the queue ther although the server hands out records in order, the records are delivered asynchronously to consumers, so they m means the ordering of the records is lost in the presence of parallel consumption. Messaging systems often work only one process to consume from a queue, but of course this means that there is no parallelism in processing.

Kafka does it better. By having a notion of parallelism—the partition—within the topics, Kafka is able to provide bot processes. This is achieved by assigning the partitions in the topic to the consumers in the consumer group so th group. By doing this we ensure that the consumer is the only reader of that partition and consumes the data in ord many consumer instances. Note however that there cannot be more consumer instances in a consumer group tha

Kafka as a Storage System

Any message queue that allows publishing messages decoupled from consuming them is effectively acting as a s
Kafka is that it is a very good storage system.

Data written to Kafka is written to disk and replicated for fault-tolerance. Kafka allows producers to wait on ackno
replicated and guaranteed to persist even if the server written to fails.

The disk structures Kafka uses scale well—Kafka will perform the same whether you have 50 KB or 50 TB of persis

As a result of taking storage seriously and allowing the clients to control their read position, you can think of Kafka
high-performance, low-latency commit log storage, replication, and propagation.

For details about the Kafka's commit log storage and replication design, please read this page.

Kafka for Stream Processing

It isn't enough to just read, write, and store streams of data, the purpose is to enable real-time processing of strear

In Kafka a stream processor is anything that takes continual streams of data from input topics, performs some pr
output topics.

For example, a retail application might take in input streams of sales and shipments, and output a stream of reord

It is possible to do simple processing directly using the producer and consumer APIs. However for more complex
allows building applications that do non-trivial processing that compute aggregations off of streams or join strean

This facility helps solve the hard problems this type of application faces: handling out-of-order data, reprocessing

The streams API builds on the core primitives Kafka provides: it uses the producer and consumer APIs for input, u
mechanism for fault tolerance among the stream processor instances.

Putting the Pieces Together

This combination of messaging, storage, and stream processing may seem unusual but it is essential to Kafka's rc

A distributed file system like HDFS allows storing static files for batch processing. Effectively a system like this all

A traditional enterprise messaging system allows processing future messages that will arrive after you subscribe.

Kafka combines both of these capabilities, and the combination is critical both for Kafka usage as a platform for s

By combining storage and low-latency subscriptions, streaming applications can treat both past and future data th
stored data but rather than ending when it reaches the last record it can keep processing as future data arrives. Th
batch processing as well as message-driven applications.

Likewise for streaming data pipelines the combination of subscription to real-time events make it possible to use I
reliably make it possible to use it for critical data where the delivery of data must be guaranteed or for integration

down for extended periods of time for maintenance. The stream processing facilities make it possible to transforr

For more information on the guarantees, APIs, and capabilities Kafka provides see the rest of the documentation.

1.2 Use Cases

Here is a description of a few of the popular use cases for Apache Kafka®. For an overview of a number of these ε

Messaging

Kafka works well as a replacement for a more traditional message broker. Message brokers are used for a variety unprocessed messages, etc). In comparison to most messaging systems Kafka has better throughput, built-in par solution for large scale message processing applications.

In our experience messaging uses are often comparatively low-throughput, but may require low end-to-end latency provides.

In this domain Kafka is comparable to traditional messaging systems such as ActiveMQ or RabbitMQ.

Website Activity Tracking

The original use case for Kafka was to be able to rebuild a user activity tracking pipeline as a set of real-time publi or other actions users may take) is published to central topics with one topic per activity type. These feeds are ava processing, real-time monitoring, and loading into Hadoop or offline data warehousing systems for offline process

Activity tracking is often very high volume as many activity messages are generated for each user page view.

Metrics

Kafka is often used for operational monitoring data. This involves aggregating statistics from distributed applicati

Log Aggregation

Many people use Kafka as a replacement for a log aggregation solution. Log aggregation typically collects physica or HDFS perhaps) for processing. Kafka abstracts away the details of files and gives a cleaner abstraction of log o latency processing and easier support for multiple data sources and distributed data consumption. In comparison good performance, stronger durability guarantees due to replication, and much lower end-to-end latency.

Stream Processing

Many users of Kafka process data in processing pipelines consisting of multiple stages, where raw input data is c otherwise transformed into new topics for further consumption or follow-up processing. For example, a processin content from RSS feeds and publish it to an "articles" topic; further processing might normalize or deduplicate this final processing stage might attempt to recommend this content to users. Such processing pipelines create graph

0.10.0.0, a light-weight but powerful stream processing library called Kafka Streams is available in Apache Kafka t
Kafka Streams, alternative open source stream processing tools include Apache Storm and Apache Samza.

Event Sourcing

Event sourcing is a style of application design where state changes are logged as a time-ordered sequence of reco
excellent backend for an application built in this style.

Commit Log

Kafka can serve as a kind of external commit-log for a distributed system. The log helps replicate data between no
restore their data. The log compaction feature in Kafka helps support this usage. In this usage Kafka is similar to A

1.3 Quick Start

This tutorial assumes you are starting fresh and have no existing Kafka or ZooKeeper data. Since Kafka console s
Windows platforms use `bin\windows\` instead of `bin/`, and change the script extension to `.bat`.

Step 1: Download the code

Download the 2.4.0 release and un-tar it.

```
> tar -xzf kafka_2.12-2.4.0.tgz
> cd kafka_2.12-2.4.0
```

Step 2: Start the server

Kafka uses ZooKeeper so you need to first start a ZooKeeper server if you don't already have one. You can use the
single-node ZooKeeper instance.

```
> bin/zookeeper-server-start.sh config/zookeeper.properties
[2013-04-22 15:01:37,495] INFO Reading configuration from: config/zookeeper.propertie
...
```

Now start the Kafka server:

```
> bin/kafka-server-start.sh config/server.properties
[2013-04-22 15:01:47,028] INFO Verifying properties (kafka.utils.VerifiableProperties
[2013-04-22 15:01:47,051] INFO Property socket.send.buffer.bytes is overridden to 104
...
```

Step 3: Create a topic

Let's create a topic named "test" with a single partition and only one replica:

```
> bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor
```

We can now see that topic if we run the list topic command:

```
> bin/kafka-topics.sh --list --bootstrap-server localhost:9092
test
```

Alternatively, instead of manually creating topics you can also configure your brokers to auto-create topics when a

Step 4: Send some messages

Kafka comes with a command line client that will take input from a file or from standard input and send it out as m separate message.

Run the producer and then type a few messages into the console to send to the server.

```
> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
This is a message
This is another message
```

Step 5: Start a consumer

Kafka also has a command line consumer that will dump out messages to standard output.

```
> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from
This is a message
This is another message
```

If you have each of the above commands running in a different terminal then you should now be able to type mess consumer terminal.

All of the command line tools have additional options; running the command with no arguments will display usage

Step 6: Setting up a multi-broker cluster

So far we have been running against a single broker, but that's no fun. For Kafka, a single broker is just a cluster of broker instances. But just to get feel for it, let's expand our cluster to three nodes (still all on our local machine).

First we make a config file for each of the brokers (on Windows use the `copy` command instead):

```
> cp config/server.properties config/server-1.properties
> cp config/server.properties config/server-2.properties
```

Now edit these new files and set the following properties:

```
config/server-1.properties:
    broker.id=1
    listeners=PLAINTEXT://:9093
    log.dirs=/tmp/kafka-logs-1

config/server-2.properties:
    broker.id=2
    listeners=PLAINTEXT://:9094
    log.dirs=/tmp/kafka-logs-2
```

The `broker.id` property is the unique and permanent name of each node in the cluster. We have to override th the same machine and we want to keep the brokers from all trying to register on the same port or overwrite each c

We already have Zookeeper and our single node started, so we just need to start the two new nodes:

```
> bin/kafka-server-start.sh config/server-1.properties &
...
> bin/kafka-server-start.sh config/server-2.properties &
...
```

Now create a new topic with a replication factor of three:

```
> bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor
```

Okay but now that we have a cluster how can we know which broker is doing what? To see that run the "describe t

```
> bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic my-replica
Topic:my-replicated-topic   PartitionCount:1    ReplicationFactor:3 Configs:
    Topic: my-replicated-topic  Partition: 0    Leader: 1   Replicas: 1,2,0 Isr: 1,2,
```

Here is an explanation of output. The first line gives a summary of all the partitions, each additional line gives info this topic there is only one line.

- "leader" is the node responsible for all reads and writes for the given partition. Each node will be the leader for a
- "replicas" is the list of nodes that replicate the log for this partition regardless of whether they are the leader or
- "isr" is the set of "in-sync" replicas. This is the subset of the replicas list that is currently alive and caught-up to t

Note that in my example node 1 is the leader for the only partition of the topic.

We can run the same command on the original topic we created to see where it is:

```
> bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic test
Topic:test  PartitionCount:1    ReplicationFactor:1 Configs:
    Topic: test Partition: 0    Leader: 0   Replicas: 0 Isr: 0
```

So there is no surprise there—the original topic has no replicas and is on server 0, the only server in our cluster wh

Let's publish a few messages to our new topic:

```
> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic my-replicated-to
...
my test message 1
my test message 2
^C
```

Now let's consume these messages:

```
> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --from-beginning --
...
my test message 1
my test message 2
^C
```

Now let's test out fault-tolerance. Broker 1 was acting as the leader so let's kill it:

```
> ps aux | grep server-1.properties
7564 ttys002    0:15.91 /System/Library/Frameworks/JavaVM.framework/Versions/1.8/Home
> kill -9 7564
```

On Windows use:

```
> wmic process where "caption = 'java.exe' and commandline like '%server-1.properties
ProcessId
6016
> taskkill /pid 6016 /f
```

Leadership has switched to one of the followers and node 1 is no longer in the in-sync replica set:

```
> bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --topic my-replica
Topic:my-replicated-topic   PartitionCount:1    ReplicationFactor:3 Configs:
    Topic: my-replicated-topic  Partition: 0    Leader: 2   Replicas: 1,2,0 Isr: 2,0
```

But the messages are still available for consumption even though the leader that took the writes originally is down

```
> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --from-beginning --
...
my test message 1
my test message 2
^C
```

Step 7: Use Kafka Connect to import/export data

Writing data from the console and writing it back to the console is a convenient place to start, but you'll probably w
other systems. For many systems, instead of writing custom integration code you can use Kafka Connect to impor

Kafka Connect is a tool included with Kafka that imports and exports data to Kafka. It is an extensible tool that rur
with an external system. In this quickstart we'll see how to run Kafka Connect with simple connectors that import
to a file.

First, we'll start by creating some seed data to test with:

```
> echo -e "foo\nbar" > test.txt
```

Or on Windows:

```
> echo foo> test.txt
> echo bar>> test.txt
```

Next, we'll start two connectors running in *standalone* mode, which means they run in a single, local, dedicated prc
is always the configuration for the Kafka Connect process, containing common configuration such as the Kafka br
remaining configuration files each specify a connector to create. These files include a unique connector name, the
required by the connector.

```
> bin/connect-standalone.sh config/connect-standalone.properties config/connect-file-
```

These sample configuration files, included with Kafka, use the default local cluster configuration you started earlie
reads lines from an input file and produces each to a Kafka topic and the second is a sink connector that reads me
output file.

During startup you'll see a number of log messages, including some indicating that the connectors are being insta connector should start reading lines from `test.txt` and producing them to the topic `connect-test`, and tl `connect-test` and write them to the file `test.sink.txt`. We can verify the data has been delivered throu

```
> more test.sink.txt
foo
bar
```

Note that the data is being stored in the Kafka topic `connect-test`, so we can also run a console consumer tc process it):

```
> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic connect-tes
{"schema":{"type":"string","optional":false},"payload":"foo"}
{"schema":{"type":"string","optional":false},"payload":"bar"}
...
```

The connectors continue to process data, so we can add data to the file and see it move through the pipeline:

```
> echo Another line>> test.txt
```

You should see the line appear in the console consumer output and in the sink file.

Step 8: Use Kafka Streams to process data

Kafka Streams is a client library for building mission-critical real-time applications and microservices, where the in combines the simplicity of writing and deploying standard Java and Scala applications on the client side with the t applications highly scalable, elastic, fault-tolerant, distributed, and much more. This quickstart example will demoı

1.4 Ecosystem

There are a plethora of tools that integrate with Kafka outside the main distribution. The ecosystem page lists maı integration, monitoring, and deployment tools.

1.5 Upgrading From Previous Versions

Upgrading from 0.8.x, 0.9.x, 0.10.0.x, 0.10.1.x, 0.10.2.x, 0.11.0.x, 1.0.x, 1.1.x, 2.0.x c

If you are upgrading from a version prior to 2.1.x, please see the note below about the change to the schema use inter.broker.protocol.version to the latest version, it will not be possible to downgrade to a version prior to 2.1.

For a rolling upgrade:

1. Update server.properties on all brokers and add the following properties. CURRENT_KAFKA_VERSION refers t CURRENT_MESSAGE_FORMAT_VERSION refers to the message format version currently in use. If you have pı its current value. Alternatively, if you are upgrading from a version prior to 0.11.0.x, then CURRENT_MESSAGE CURRENT_KAFKA_VERSION.
    - inter.broker.protocol.version=CURRENT_KAFKA_VERSION (e.g. 0.10.0, 0.11.0, 1.0, 2.0, 2.2).

- log.message.format.version=CURRENT_MESSAGE_FORMAT_VERSION (See [potential performance impact](#) does.)

If you are upgrading from version 0.11.0.x or above, and you have not overridden the message format, then yc

- inter.broker.protocol.version=CURRENT_KAFKA_VERSION (0.11.0, 1.0, 1.1, 2.0, 2.1, 2.2, 2.3).

2. Upgrade the brokers one at a time: shut down the broker, update the code, and restart it. Once you have done that the cluster's behavior and performance meets expectations. It is still possible to downgrade at this point
3. Once the cluster's behavior and performance has been verified, bump the protocol version by editing `inter.`
4. Restart the brokers one by one for the new protocol version to take effect. Once the brokers begin using the la the cluster to an older version.
5. If you have overridden the message format version as instructed above, then you need to do one more rolling consumers have been upgraded to 0.11.0 or later, change log.message.format.version to 2.4 on each broker a which are no longer maintained, do not support the message format introduced in 0.11, so to avoid conversio newer Java clients must be used.

Additional Upgrade Notes:

1. ZooKeeper has been upgraded to 3.5.6. ZooKeeper upgrade from 3.4.X to 3.5.6 can fail if there are no snapsh where ZooKeeper 3.5.6 is trying to load an existing 3.4 data dir in which no snapshot file has been created. Fc fix is given in [ZOOKEEPER-3056](#), which is to set `snapshot.trust.empty=true` config in `zookeeper.|` in standalone cluster upgrades when using `snapshot.trust.empty=true` config. For more details abou the safe workaround of copying empty [snapshot](#) file to the 3.4 data directory, if there are no snapshot files in : refer to [ZooKeeper Upgrade FAQ](#).
2. An embedded Jetty based [AdminServer](#) added in ZooKeeper 3.5. AdminServer is enabled by default in ZooKe default in the ZooKeeper config (`zookeeper.properties`) provided by the Apache Kafka distribution. Ma `admin.enableServer=false` if you wish to disable the AdminServer. Please refer [AdminServer config](#) tc

Notable changes in 2.4.0

- A new Admin API has been added for partition reassignments. Due to changing the way Kafka propagates reas failure edge cases while upgrading to the new version. It is not recommended to start reassignments while upg
- ZooKeeper has been upgraded from 3.4.14 to 3.5.6. TLS and dynamic reconfiguration are supported by the new
- The `bin/kafka-preferred-replica-election.sh` command line tool has been deprecated. It has be
- The methods `electPreferredLeaders` in the Java `AdminClient` class have been deprecated in favor
- Scala code leveraging the `NewTopic(String, int, short)` constructor with literal values will need to e
- The argument in the constructor `GroupAuthorizationException(String)` is now used to specify an e authorization. This was done for consistency with other exception types and to avoid potential misuse. The cor was previously used for a single unauthorized topic was changed similarly.
- The internal `PartitionAssignor` interface has been deprecated and replaced with a new `ConsumerPar` are slightly different between the two interfaces. Users implementing a custom PartitionAssignor should migra
- The `DefaultPartitioner` now uses a sticky partitioning strategy. This means that records for specific top partition until the batch is ready to be sent. When a new batch is created, a new partition is chosen. This decrea

records across partitions in edge cases. Generally users will not be impacted, but this difference may be notice amount of time.

- The blocking `KafkaConsumer#committed` methods have been extended to allow a list of partitions as inpu request/response iterations between clients and brokers fetching for the committed offsets for the consumer g recommend users to make their code changes to leverage the new methods (details can be found in KIP-520).
- We've introduced a new `INVALID_RECORD` error in the produce response to distinguish from the `CORRUPT_` records were sent as part of a single request to the broker and one or more of the records failed the validation d errors, null key for log compacted topics, etc), the whole batch would be rejected with the same and misleading see the corresponding exception from either the future object of `RecordMetadata` returned from the `senc` `Callback#onCompletion(RecordMetadata metadata, Exception exception)` Now with the new producer callers would be better informed about the root cause why their sent records were failed.
- We are introducing incremental cooperative rebalancing to the clients' group protocol, which allows consumers the end revoke only those which must be migrated to another consumer for overall cluster balance. The `Cons` `RebalanceProtocol` that is commonly supported by all of the consumer's supported assignors. You can u your own custom cooperative assignor. To do so you must implement the `ConsumerPartitionAssignor` the list returned by `ConsumerPartitionAssignor#supportedProtocols`. Your custom assignor can th `Subscription` to give partitions back to their previous owners whenever possible. Note that when a partitio from the new assignment until it has been revoked from its original owner. Any consumer that has to revoke a p partition to safely be assigned to its new owner. See the ConsumerPartitionAssignor RebalanceProtocol javadc To upgrade from the old (eager) protocol, which always revokes all partitions before rebalancing, to cooperative clients on the same `ConsumerPartitionAssignor` that supports the cooperative protocol. This can be do `CooperativeStickyAssignor` for the example: during the first one, add "cooperative-sticky" to the list of previous assignor -- note that if previously using the default, you must include that explicitly as well). You then b members have the "cooperative-sticky" among their supported assignors, remove the other assignor(s) and per support only the cooperative protocol. For further details on the cooperative rebalancing protocol and upgrade
- There are some behavioral changes to the `ConsumerRebalanceListener`, as well as a new API. Exceptio be swallowed, and will instead be re-thrown all the way up to the `Consumer.poll()` call. The `onPartiti` abnormal circumstances where a consumer may have lost ownership of its partitions (such as a missed rebala existing `onPartitionsRevoked` API to align with previous behavior. Note however that `onPartitionsL` This means that no callback will be invoked at the beginning of the first rebalance of a new consumer joining th The semantics of the `ConsumerRebalanceListener's` callbacks are further changed when following the `onPartitionsLost`, `onPartitionsRevoked` will also never be called when the set of revoked partitior of a rebalance, and only on the set of partitions that are being moved to another consumer. The `onPartitio` empty set of partitions, as a way to notify users of a rebalance event (this is true for both cooperative and eage ConsumerRebalanceListener javadocs.

Upgrading from 0.8.x, 0.9.x, 0.10.0.x, 0.10.1.x, 0.10.2.x, 0.11.0.x, 1.0.x, 1.1.x, 2.0.x c

If you are upgrading from a version prior to 2.1.x, please see the note below about the change to the schema use inter.broker.protocol.version to the latest version, it will not be possible to downgrade to a version prior to 2.1.

For a rolling upgrade:

1. Update server.properties on all brokers and add the following properties. CURRENT_KAFKA_VERSION refers t CURRENT_MESSAGE_FORMAT_VERSION refers to the message format version currently in use. If you have pı its current value. Alternatively, if you are upgrading from a version prior to 0.11.0.x, then CURRENT_MESSAGE CURRENT_KAFKA_VERSION.
   - inter.broker.protocol.version=CURRENT_KAFKA_VERSION (e.g. 0.8.2, 0.9.0, 0.10.0, 0.10.1, 0.10.2, 0.11.0, 1.
   - log.message.format.version=CURRENT_MESSAGE_FORMAT_VERSION (See potential performance impact does.)

   If you are upgrading from 0.11.0.x, 1.0.x, 1.1.x, 2.0.x, or 2.1.x, and you have not overridden the message forma
   - inter.broker.protocol.version=CURRENT_KAFKA_VERSION (0.11.0, 1.0, 1.1, 2.0, 2.1, 2.2).
2. Upgrade the brokers one at a time: shut down the broker, update the code, and restart it. Once you have done that the cluster's behavior and performance meets expectations. It is still possible to downgrade at this point
3. Once the cluster's behavior and performance has been verified, bump the protocol version by editing `inter.`
4. Restart the brokers one by one for the new protocol version to take effect. Once the brokers begin using the la the cluster to an older version.
5. If you have overridden the message format version as instructed above, then you need to do one more rolling consumers have been upgraded to 0.11.0 or later, change log.message.format.version to 2.3 on each broker a which are no longer maintained, do not support the message format introduced in 0.11, so to avoid conversio newer Java clients must be used.

Notable changes in 2.3.0

- We are introducing a new rebalancing protocol for Kafka Connect based on incremental cooperative rebalancin a rebalancing phase between Connect workers. Instead, only the tasks that need to be exchanged between wor new Connect protocol is enabled by default beginning with 2.3.0. For more details on how it works and how to e cooperative rebalancing design.
- We are introducing static membership towards consumer user. This feature reduces unnecessary rebalances d details on how to use it, checkout static membership design.
- Kafka Streams DSL switches its used store types. While this change is mainly transparent to users, there are sc Streams upgrade section for more details.
- Kafka Streams 2.3.0 requires 0.11 message format or higher and does not work with older message format.

Upgrading from 0.8.x, 0.9.x, 0.10.0.x, 0.10.1.x, 0.10.2.x, 0.11.0.x, 1.0.x, 1.1.x, 2.0.x c

If you are upgrading from a version prior to 2.1.x, please see the note below about the change to the schema use inter.broker.protocol.version to the latest version, it will not be possible to downgrade to a version prior to 2.1.

For a rolling upgrade:

1. Update server.properties on all brokers and add the following properties. CURRENT_KAFKA_VERSION refers t CURRENT_MESSAGE_FORMAT_VERSION refers to the message format version currently in use. If you have pı

its current value. Alternatively, if you are upgrading from a version prior to 0.11.0.x, then CURRENT_MESSAGE_
CURRENT_KAFKA_VERSION.

- inter.broker.protocol.version=CURRENT_KAFKA_VERSION (e.g. 0.8.2, 0.9.0, 0.10.0, 0.10.1, 0.10.2, 0.11.0, 1.
- log.message.format.version=CURRENT_MESSAGE_FORMAT_VERSION (See potential performance impact
does.)

If you are upgrading from 0.11.0.x, 1.0.x, 1.1.x, or 2.0.x and you have not overridden the message format, then

- inter.broker.protocol.version=CURRENT_KAFKA_VERSION (0.11.0, 1.0, 1.1, 2.0).

2. Upgrade the brokers one at a time: shut down the broker, update the code, and restart it. Once you have done that the cluster's behavior and performance meets expectations. It is still possible to downgrade at this point
3. Once the cluster's behavior and performance has been verified, bump the protocol version by editing `inter.`
4. Restart the brokers one by one for the new protocol version to take effect. Once the brokers begin using the la the cluster to an older version.
5. If you have overridden the message format version as instructed above, then you need to do one more rolling consumers have been upgraded to 0.11.0 or later, change log.message.format.version to 2.2 on each broker a which are no longer maintained, do not support the message format introduced in 0.11, so to avoid conversio newer Java clients must be used.

Notable changes in 2.2.1

- Kafka Streams 2.2.1 requires 0.11 message format or higher and does not work with older message format.

Notable changes in 2.2.0

- The default consumer group id has been changed from the empty string (`""`) to `null`. Consumers who us and fetch or commit offsets. The empty string as consumer group id is deprecated but will be supported until a group id will now have to explicitly provide it as part of their consumer config. For more information see KIP-28
- The `bin/kafka-topics.sh` command line tool is now able to connect directly to brokers with `--bootst` option is still available for now. Please read KIP-377 for more information.
- Kafka Streams depends on a newer version of RocksDBs that requires MacOS 10.13 or higher.

Upgrading from 0.8.x, 0.9.x, 0.10.0.x, 0.10.1.x, 0.10.2.x, 0.11.0.x, 1.0.x, 1.1.x, or 2.0.

Note that 2.1.x contains a change to the internal schema used to store consumer offsets. Once the upgrade is co See the rolling upgrade notes below for more detail.

For a rolling upgrade:

1. Update server.properties on all brokers and add the following properties. CURRENT_KAFKA_VERSION refers t CURRENT_MESSAGE_FORMAT_VERSION refers to the message format version currently in use. If you have pr its current value. Alternatively, if you are upgrading from a version prior to 0.11.0.x, then CURRENT_MESSAGE_ CURRENT_KAFKA_VERSION.

- inter.broker.protocol.version=CURRENT_KAFKA_VERSION (e.g. 0.8.2, 0.9.0, 0.10.0, 0.10.1, 0.10.2, 0.11.0, 1.
- log.message.format.version=CURRENT_MESSAGE_FORMAT_VERSION (See potential performance impact does.)

If you are upgrading from 0.11.0.x, 1.0.x, 1.1.x, or 2.0.x and you have not overridden the message format, then

- inter.broker.protocol.version=CURRENT_KAFKA_VERSION (0.11.0, 1.0, 1.1, 2.0).

2. Upgrade the brokers one at a time: shut down the broker, update the code, and restart it. Once you have done that the cluster's behavior and performance meets expectations. It is still possible to downgrade at this point
3. Once the cluster's behavior and performance has been verified, bump the protocol version by editing `inter.`
4. Restart the brokers one by one for the new protocol version to take effect. Once the brokers begin using the la the cluster to an older version.
5. If you have overridden the message format version as instructed above, then you need to do one more rolling consumers have been upgraded to 0.11.0 or later, change log.message.format.version to 2.1 on each broker a which are no longer maintained, do not support the message format introduced in 0.11, so to avoid conversio newer Java clients must be used.

Additional Upgrade Notes:

1. Offset expiration semantics has slightly changed in this version. According to the new semantics, offsets of p subscribed to the corresponding topic and is still active (has active consumers). If group becomes empty all i the one set by broker) has passed (unless the group becomes active again). Offsets associated with standalc will be removed after default offset retention period (or the one set by broker) has passed since their last com
2. The default for console consumer's `enable.auto.commit` property when no `group.id` is provided is coordinator cache as the auto-generated group is not likely to be used by other consumers.
3. The default value for the producer's `retries` config was changed to `Integer.MAX_VALUE`, as we intro bound on the total time between sending a record and receiving acknowledgement from the broker. By defaul
4. By default, MirrorMaker now overrides `delivery.timeout.ms` to `Integer.MAX_VALUE` when configu in order to fail faster, you will instead need to override `delivery.timeout.ms`.
5. The `ListGroup` API now expects, as a recommended alternative, `Describe Group` access to the grou `Cluster` access is still supported for backward compatibility, using it for this API is not advised.
6. KIP-336 deprecates the ExtendedSerializer and ExtendedDeserializer interfaces and propagates the usage of ExtendedDeserializer were introduced with KIP-82 to provide record headers for serializers and deserializers i interfaces as Java 7 support has been dropped since.

Notable changes in 2.1.0

- Jetty has been upgraded to 9.4.12, which excludes TLS_RSA_* ciphers by default because they do not support https://github.com/eclipse/jetty.project/issues/2807 for more information.
- Unclean leader election is automatically enabled by the controller when `unclean.leader.election.enab` override.
- The `AdminClient` has added a method `AdminClient#metrics()`. Now any application using the `Adr` metrics captured from the `AdminClient`. For more information see KIP-324

- Kafka now supports Zstandard compression from KIP-110. You must upgrade the broker as well as clients to m
  from topics which use Zstandard compression, so you should not enable it for a topic until all downstream con

Upgrading from 0.8.x, 0.9.x, 0.10.0.x, 0.10.1.x, 0.10.2.x, 0.11.0.x, 1.0.x, or 1.1.x to 2.

Kafka 2.0.0 introduces wire protocol changes. By following the recommended rolling upgrade plan below, you guai
the notable changes in 2.0.0 before upgrading.

For a rolling upgrade:

1. Update server.properties on all brokers and add the following properties. CURRENT_KAFKA_VERSION refers t
   CURRENT_MESSAGE_FORMAT_VERSION refers to the message format version currently in use. If you have pı
   its current value. Alternatively, if you are upgrading from a version prior to 0.11.0.x, then CURRENT_MESSAGE
   CURRENT_KAFKA_VERSION.
   - inter.broker.protocol.version=CURRENT_KAFKA_VERSION (e.g. 0.8.2, 0.9.0, 0.10.0, 0.10.1, 0.10.2, 0.11.0, 1.
   - log.message.format.version=CURRENT_MESSAGE_FORMAT_VERSION (See potential performance impact
     does.)

   If you are upgrading from 0.11.0.x, 1.0.x, or 1.1.x and you have not overridden the message format, then you c

   - inter.broker.protocol.version=CURRENT_KAFKA_VERSION (0.11.0, 1.0, 1.1).
2. Upgrade the brokers one at a time: shut down the broker, update the code, and restart it.
3. Once the entire cluster is upgraded, bump the protocol version by editing `inter.broker.protocol.vers`
4. Restart the brokers one by one for the new protocol version to take effect.
5. If you have overridden the message format version as instructed above, then you need to do one more rolling
   consumers have been upgraded to 0.11.0 or later, change log.message.format.version to 2.0 on each broker a
   does not support the new message format introduced in 0.11, so to avoid the performance cost of down-conv
   Java consumer must be used.

Additional Upgrade Notes:

1. If you are willing to accept downtime, you can simply take all the brokers down, update the code and start the
2. Bumping the protocol version and restarting can be done any time after the brokers are upgraded. It does not
   version.
3. If you are using Java8 method references in your Kafka Streams code you might need to update your code to
   not work.
4. ACLs should not be added to prefixed resources, (added in KIP-290), until all brokers in the cluster have been

   NOTE: any prefixed ACLs added to a cluster, even after the cluster is fully upgraded, will be ignored should the

Notable changes in 2.0.0

- KIP-186 increases the default offset retention time from 1 day to 7 days. This makes it less likely to "lose" offse
  active set of offsets and therefore can increase memory usage on the broker. Note that the console consumer

of a large number of offsets which this change will now preserve for 7 days instead of 1. You can preserve the
`offsets.retention.minutes` to 1440.

- Support for Java 7 has been dropped, Java 8 is now the minimum version required.
- The default value for `ssl.endpoint.identification.algorithm` was changed to `https`, which per possible otherwise). Set `ssl.endpoint.identification.algorithm` to an empty string to restore the
- KAFKA-5674 extends the lower interval of `max.connections.per.ip` minimum to zero and therefore allo
- KIP-272 added API version tag to the metric `kafka.network:type=RequestMetrics,name=RequestsP {Produce|FetchConsumer|FetchFollower|...}`. This metric now becomes `kafka.network:type= {Produce|FetchConsumer|FetchFollower|...},version={0|1|2|3|...}`. This will impact JMX n total count for a specific request type, the tool needs to be updated to aggregate across different versions.
- KIP-225 changed the metric "records.lag" to use tags for topic and partition. The original version with the name
- The Scala consumers, which have been deprecated since 0.11.0.0, have been removed. The Java consumer ha consumers in 1.1.0 (and older) will continue to work even if the brokers are upgraded to 2.0.0.
- The Scala producers, which have been deprecated since 0.10.0.0, have been removed. The Java producer has b behaviour of the default partitioner in the Java producer differs from the default partitioner in the Scala produce partitioner that retains the previous behaviour. Note that the Scala producers in 1.1.0 (and older) will continue t
- MirrorMaker and ConsoleConsumer no longer support the Scala consumer, they always use the Java consume
- The ConsoleProducer no longer supports the Scala producer, it always uses the Java producer.
- A number of deprecated tools that rely on the Scala clients have been removed: ReplayLogProducer, SimpleCor ImportZkOffsets, UpdateOffsetsInZK, VerifyConsumerRebalance.
- The deprecated kafka.tools.ProducerPerformance has been removed, please use org.apache.kafka.tools.Produ
- New Kafka Streams configuration parameter `upgrade.from` added that allows rolling bounce upgrade from
- KIP-284 changed the retention time for Kafka Streams repartition topics by setting its default value to `Long.M`
- Updated `ProcessorStateManager` APIs in Kafka Streams for registering state stores to the processor top
- In earlier releases, Connect's worker configuration required the `internal.key.converter` and `interna` required and default to the JSON converter. You may safely remove these properties from your Connect standa `internal.key.converter=org.apache.kafka.connect.json.JsonConverter` `internal.key` `internal.value.converter=org.apache.kafka.connect.json.JsonConverter` `internal.va`
- KIP-266 adds a new consumer configuration `default.api.timeout.ms` to specify the default timeout to adds overloads for such blocking APIs to support specifying a specific timeout to use for each of them instead `default.api.timeout.ms`. In particular, a new `poll(Duration)` API has been added which does not API has been deprecated and will be removed in a future version. Overloads have also been added for other `Ka` `listTopics`, `offsetsForTimes`, `beginningOffsets`, `endOffsets` and `close` that take in a
- Also as part of KIP-266, the default value of `request.timeout.ms` has been changed to 30 seconds. The p maximum time that a rebalance would take. Now we treat the JoinGroup request in the rebalance as a special the request timeout. All other request types use the timeout defined by `request.timeout.ms`
- The internal method `kafka.admin.AdminClient.deleteRecordsBefore` has been removed. Users are `org.apache.kafka.clients.admin.AdminClient.deleteRecords`.
- The AclCommand tool `--producer` convenience option uses the KIP-277 finer grained ACL on the given top
- KIP-176 removes the `--new-consumer` option for all consumer based tools. This option is redundant since defined.
- KIP-290 adds the ability to define ACLs on prefixed resources, e.g. any topic starting with 'foo'.

- [KIP-283](#) improves message down-conversion handling on Kafka broker, which has typically been a memory-inte operation becomes less memory intensive by down-converting chunks of partition data at a time which helps p improvement, there is a change in `FetchResponse` protocol behavior where the broker could send an overs invalid offset. Such oversized messages must be ignored by consumer clients, as is done by `KafkaConsume`

  KIP-283 also adds new topic and broker configurations `message.downconversion.enable` and `log.me` whether down-conversion is enabled. When disabled, broker does not perform any down-conversion and instea

- Dynamic broker configuration options can be stored in ZooKeeper using kafka-configs.sh before brokers are st server.properties as all password configs may be stored encrypted in ZooKeeper.
- ZooKeeper hosts are now re-resolved if connection attempt fails. But if your ZooKeeper host names resolve to may need to increase the connection timeout `zookeeper.connection.timeout.ms`.

New Protocol Versions

- KIP-279: OffsetsForLeaderEpochResponse v1 introduces a partition-level `leader_epoch` field.
- KIP-219: Bump up the protocol versions of non-cluster action requests and responses that are throttled on quo
- KIP-290: Bump up the protocol versions ACL create, describe and delete requests and responses.

Upgrading a 1.1 Kafka Streams Application

- Upgrading your Streams application from 1.1 to 2.0 does not require a broker upgrade. A Kafka Streams 2.0 ap brokers (it is not possible to connect to 0.10.0 brokers though).
- Note that in 2.0 we have removed the public APIs that are deprecated prior to 1.0; users leveraging on those de Streams API changes in 2.0.0 for more details.

Upgrading from 0.8.x, 0.9.x, 0.10.0.x, 0.10.1.x, 0.10.2.x, 0.11.0.x, or 1.0.x to 1.1.x

Kafka 1.1.0 introduces wire protocol changes. By following the recommended rolling upgrade plan below, you gua the notable changes in 1.1.0 before upgrading.

For a rolling upgrade:

1. Update server.properties on all brokers and add the following properties. CURRENT_KAFKA_VERSION refers t CURRENT_MESSAGE_FORMAT_VERSION refers to the message format version currently in use. If you have p its current value. Alternatively, if you are upgrading from a version prior to 0.11.0.x, then CURRENT_MESSAGE CURRENT_KAFKA_VERSION.
   - inter.broker.protocol.version=CURRENT_KAFKA_VERSION (e.g. 0.8.2, 0.9.0, 0.10.0, 0.10.1, 0.10.2, 0.11.0, 1.
   - log.message.format.version=CURRENT_MESSAGE_FORMAT_VERSION (See potential performance impact does.)

   If you are upgrading from 0.11.0.x or 1.0.x and you have not overridden the message format, then you only ne

   - inter.broker.protocol.version=CURRENT_KAFKA_VERSION (0.11.0 or 1.0).

2. Upgrade the brokers one at a time: shut down the broker, update the code, and restart it.
3. Once the entire cluster is upgraded, bump the protocol version by editing `inter.broker.protocol.vers`
4. Restart the brokers one by one for the new protocol version to take effect.
5. If you have overridden the message format version as instructed above, then you need to do one more rolling consumers have been upgraded to 0.11.0 or later, change log.message.format.version to 1.1 on each broker a does not support the new message format introduced in 0.11, so to avoid the performance cost of down-conv Java consumer must be used.

Additional Upgrade Notes:

1. If you are willing to accept downtime, you can simply take all the brokers down, update the code and start the
2. Bumping the protocol version and restarting can be done any time after the brokers are upgraded. It does not version.
3. If you are using Java8 method references in your Kafka Streams code you might need to update your code to not work.

Notable changes in 1.1.1

- New Kafka Streams configuration parameter `upgrade.from` added that allows rolling bounce upgrade from
- See the **Kafka Streams upgrade guide** for details about this new config.

Notable changes in 1.1.0

- The kafka artifact in Maven no longer depends on log4j or slf4j-log4j12. Similarly to the kafka-clients artifact, us appropriate slf4j module (slf4j-log4j12, logback, etc.). The release tarball still includes log4j and slf4j-log4j12.
- KIP-225 changed the metric "records.lag" to use tags for topic and partition. The original version with the name removed in 2.0.0.
- Kafka Streams is more robust against broker communication errors. Instead of stopping the Kafka Streams clie reconnect to the cluster. Using the new `AdminClient` you have better control of how often Kafka Streams re coded retries as in older version).
- Kafka Streams rebalance time was reduced further making Kafka Streams more responsive.
- Kafka Connect now supports message headers in both sink and source connectors, and to manipulate them via explicitly use them. A new `HeaderConverter` is introduced to control how headers are (de)serialized, and t representations of values.
- kafka.tools.DumpLogSegments now automatically sets deep-iteration option if print-data-log is enabled explicit

New Protocol Versions

- KIP-226 introduced DescribeConfigs Request/Response v1.
- KIP-227 introduced Fetch Request/Response v7.

Upgrading a 1.0 Kafka Streams Application

- Upgrading your Streams application from 1.0 to 1.1 does not require a broker upgrade. A Kafka Streams 1.1 apı not possible to connect to 0.10.0 brokers though).
- See Streams API changes in 1.1.0 for more details.

Upgrading from 0.8.x, 0.9.x, 0.10.0.x, 0.10.1.x, 0.10.2.x or 0.11.0.x to 1.0.0

Kafka 1.0.0 introduces wire protocol changes. By following the recommended rolling upgrade plan below, you guaı the notable changes in 1.0.0 before upgrading.

For a rolling upgrade:

1. Update server.properties on all brokers and add the following properties. CURRENT_KAFKA_VERSION refers t CURRENT_MESSAGE_FORMAT_VERSION refers to the message format version currently in use. If you have pı its current value. Alternatively, if you are upgrading from a version prior to 0.11.0.x, then CURRENT_MESSAGE CURRENT_KAFKA_VERSION.
    - inter.broker.protocol.version=CURRENT_KAFKA_VERSION (e.g. 0.8.2, 0.9.0, 0.10.0, 0.10.1, 0.10.2, 0.11.0).
    - log.message.format.version=CURRENT_MESSAGE_FORMAT_VERSION (See potential performance impact does.)

    If you are upgrading from 0.11.0.x and you have not overridden the message format, you must set both the m 0.11.0.

    - inter.broker.protocol.version=0.11.0
    - log.message.format.version=0.11.0
2. Upgrade the brokers one at a time: shut down the broker, update the code, and restart it.
3. Once the entire cluster is upgraded, bump the protocol version by editing `inter.broker.protocol.vers`
4. Restart the brokers one by one for the new protocol version to take effect.
5. If you have overridden the message format version as instructed above, then you need to do one more rolling consumers have been upgraded to 0.11.0 or later, change log.message.format.version to 1.0 on each broker a log.message.format.version is set to 0.11.0, you can update the config and skip the rolling restart. Note that t format introduced in 0.11, so to avoid the performance cost of down-conversion (or to take advantage of exa

Additional Upgrade Notes:

1. If you are willing to accept downtime, you can simply take all the brokers down, update the code and start the
2. Bumping the protocol version and restarting can be done any time after the brokers are upgraded. It does not version.

Notable changes in 1.0.2

- New Kafka Streams configuration parameter `upgrade.from` added that allows rolling bounce upgrade from
- See the **Kafka Streams upgrade guide** for details about this new config.

Notable changes in 1.0.1

- Restored binary compatibility of AdminClient's Options classes (e.g. CreateTopicsOptions, DeleteTopicsOptions broken inadvertently in 1.0.0.

Notable changes in 1.0.0

- Topic deletion is now enabled by default, since the functionality is now stable. Users who wish to to retain the p `delete.topic.enable` to `false` . Keep in mind that topic deletion removes data and the operation is nc
- For topics that support timestamp search if no offset can be found for a partition, that partition is now includec partition was not included in the map. This change was made to make the search behavior consistent with the
- If the `inter.broker.protocol.version` is 1.0 or later, a broker will now stay online to serve replicas on directory may become offline due to IOException caused by hardware failure. Users need to monitor the per-bro there is offline log directory.
- Added KafkaStorageException which is a retriable exception. KafkaStorageException will be converted to NotL FetchRequest or ProducerRequest does not support KafkaStorageException.
- -XX:+DisableExplicitGC was replaced by -XX:+ExplicitGCInvokesConcurrent in the default JVM settings. This he memory by direct buffers in some cases.
- The overridden `handleError` method implementations have been removed from the following deprecated c `GroupCoordinatorRequest` , `OffsetCommitRequest` , `OffsetFetchRequest` , `OffsetRequest` was only intended for use on the broker, but it is no longer in use and the implementations have not been maint compatibility.
- The Java clients and tools now accept any string as a client-id.
- The deprecated tool `kafka-consumer-offset-checker.sh` has been removed. Use `kafka-consumer`
- SimpleAclAuthorizer now logs access denials to the authorizer log by default.
- Authentication failures are now reported to clients as one of the subclasses of `AuthenticationException` authentication.
- Custom `SaslServer` implementations may throw `SaslAuthenticationException` to provide an erro authentication failure. Implementors should take care not to include any security-critical information in the exce clients.
- The `app-info` mbean registered with JMX to provide version and commit id will be deprecated and replacec
- Kafka metrics may now contain non-numeric values. `org.apache.kafka.common.Metric#value()` has the probability of breaking users who read the value of every client metric (via a `MetricsReporter` implem `org.apache.kafka.common.Metric#metricValue()` can be used to retrieve numeric and non-numeri
- Every Kafka rate metric now has a corresponding cumulative count metric with the suffix `-total` to simplify `rate` has a corresponding metric named `records-consumed-total` .
- Mx4j will only be enabled if the system property `kafka_mx4jenable` is set to `true` . Due to a logic invers `kafka_mx4jenable` was set to `true` .
- The package `org.apache.kafka.common.security.auth` in the clients jar has been made public and a located in this package have been moved elsewhere.
- When using an Authorizer and a user doesn't have required permissions on a topic, the broker will return TOPIC existence on broker. If the user have required permissions and the topic doesn't exists, then the UNKNOWN_TO
- config/consumer.properties file updated to use new consumer config properties.

New Protocol Versions

- KIP-112: LeaderAndIsrRequest v1 introduces a partition-level `is_new` field.
- KIP-112: UpdateMetadataRequest v4 introduces a partition-level `offline_replicas` field.
- KIP-112: MetadataResponse v5 introduces a partition-level `offline_replicas` field.
- KIP-112: ProduceResponse v4 introduces error code for KafkaStorageException.
- KIP-112: FetchResponse v6 introduces error code for KafkaStorageException.
- KIP-152: SaslAuthenticate request has been added to enable reporting of authentication failures. This request v

Upgrading a 0.11.0 Kafka Streams Application

- Upgrading your Streams application from 0.11.0 to 1.0 does not require a broker upgrade. A Kafka Streams 1.0 not possible to connect to 0.10.0 brokers though). However, Kafka Streams 1.0 requires 0.10 message format c
- If you are monitoring on streams metrics, you will need make some changes to the metrics names in your repor was changed.
- There are a few public APIs including `ProcessorContext#schedule()`, `Processor#punctuate()` a deprecated by new APIs. We recommend making corresponding code changes, which should be very minor sin
- See Streams API changes in 1.0.0 for more details.

Upgrading a 0.10.2 Kafka Streams Application

- Upgrading your Streams application from 0.10.2 to 1.0 does not require a broker upgrade. A Kafka Streams 1.0 is not possible to connect to 0.10.0 brokers though).
- If you are monitoring on streams metrics, you will need make some changes to the metrics names in your repor was changed.
- There are a few public APIs including `ProcessorContext#schedule()`, `Processor#punctuate()` a deprecated by new APIs. We recommend making corresponding code changes, which should be very minor sin
- If you specify customized `key.serde`, `value.serde` and `timestamp.extractor` in configs, it is rec configs are deprecated.
- See Streams API changes in 0.11.0 for more details.

Upgrading a 0.10.1 Kafka Streams Application

- Upgrading your Streams application from 0.10.1 to 1.0 does not require a broker upgrade. A Kafka Streams 1.0 is not possible to connect to 0.10.0 brokers though).
- You need to recompile your code. Just swapping the Kafka Streams library jar file will not work and will break yo
- If you are monitoring on streams metrics, you will need make some changes to the metrics names in your repor was changed.
- There are a few public APIs including `ProcessorContext#schedule()`, `Processor#punctuate()` a deprecated by new APIs. We recommend making corresponding code changes, which should be very minor sin
- If you specify customized `key.serde`, `value.serde` and `timestamp.extractor` in configs, it is rec configs are deprecated.

- If you use a custom (i.e., user implemented) timestamp extractor, you will need to update this code, because th
- If you register custom metrics, you will need to update this code, because the `StreamsMetric` interface wa
- See Streams API changes in 1.0.0, Streams API changes in 0.11.0 and Streams API changes in 0.10.2 for more

Upgrading a 0.10.0 Kafka Streams Application

- Upgrading your Streams application from 0.10.0 to 1.0 does require a broker upgrade because a Kafka Streams brokers.
- There are couple of API changes, that are not backward compatible (cf. Streams API changes in 1.0.0, Streams Streams API changes in 0.10.1 for more details). Thus, you need to update and recompile your code. Just swap your application.
- Upgrading from 0.10.0.x to 1.0.2 requires two rolling bounces with config `upgrade.from="0.10.0"` set fo upgrade is also possible.
  - prepare your application instances for a rolling bounce and make sure that config `upgrade.from` is set t
  - bounce each instance of your application once
  - prepare your newly deployed 1.0.2 application instances for a second round of rolling bounces; make sure tc
  - bounce each instance of your application once more to complete the upgrade
- Upgrading from 0.10.0.x to 1.0.0 or 1.0.1 requires an offline upgrade (rolling bounce upgrade is not supported)
  - stop all old (0.10.0.x) application instances
  - update your code and swap old code and jar file with new code and new jar file
  - restart all new (1.0.0 or 1.0.1) application instances

Upgrading from 0.8.x, 0.9.x, 0.10.0.x, 0.10.1.x or 0.10.2.x to 0.11.0.0

Kafka 0.11.0.0 introduces a new message format version as well as wire protocol changes. By following the recon during the upgrade. However, please review the notable changes in 0.11.0.0 before upgrading.

Starting with version 0.10.2, Java clients (producer and consumer) have acquired the ability to communicate with newer brokers. However, if your brokers are older than 0.10.0, you must upgrade all the brokers in the Kafka cluste 0.8.x and newer clients.

For a rolling upgrade:

1. Update server.properties on all brokers and add the following properties. CURRENT_KAFKA_VERSION refers t CURRENT_MESSAGE_FORMAT_VERSION refers to the current message format version currently in use. If you CURRENT_MESSAGE_FORMAT_VERSION should be set to match CURRENT_KAFKA_VERSION.
   - inter.broker.protocol.version=CURRENT_KAFKA_VERSION (e.g. 0.8.2, 0.9.0, 0.10.0, 0.10.1 or 0.10.2).
   - log.message.format.version=CURRENT_MESSAGE_FORMAT_VERSION (See potential performance impact does.)
2. Upgrade the brokers one at a time: shut down the broker, update the code, and restart it.

3. Once the entire cluster is upgraded, bump the protocol version by editing `inter.broker.protocol.vers` `log.message.format.version` yet.
4. Restart the brokers one by one for the new protocol version to take effect.
5. Once all (or most) consumers have been upgraded to 0.11.0 or later, then change log.message.format.versior the older Scala consumer does not support the new message format, so to avoid the performance cost of do the new Java consumer must be used.

Additional Upgrade Notes:

1. If you are willing to accept downtime, you can simply take all the brokers down, update the code and start the
2. Bumping the protocol version and restarting can be done any time after the brokers are upgraded. It does not version.
3. It is also possible to enable the 0.11.0 message format on individual topics using the topic admin tool (`bin/` `log.message.format.version` .
4. If you are upgrading from a version prior to 0.10.0, it is NOT necessary to first update the message format to (

Upgrading a 0.10.2 Kafka Streams Application

- Upgrading your Streams application from 0.10.2 to 0.11.0 does not require a broker upgrade. A Kafka Streams ((it is not possible to connect to 0.10.0 brokers though).
- If you specify customized `key.serde` , `value.serde` and `timestamp.extractor` in configs, it is rec configs are deprecated.
- See [Streams API changes in 0.11.0] for more details.

Upgrading a 0.10.1 Kafka Streams Application

- Upgrading your Streams application from 0.10.1 to 0.11.0 does not require a broker upgrade. A Kafka Streams ((it is not possible to connect to 0.10.0 brokers though).
- You need to recompile your code. Just swapping the Kafka Streams library jar file will not work and will break yo
- If you specify customized `key.serde` , `value.serde` and `timestamp.extractor` in configs, it is rec configs are deprecated.
- If you use a custom (i.e., user implemented) timestamp extractor, you will need to update this code, because th
- If you register custom metrics, you will need to update this code, because the `StreamsMetric` interface wa
- See [Streams API changes in 0.11.0] and [Streams API changes in 0.10.2] for more details.

Upgrading a 0.10.0 Kafka Streams Application

- Upgrading your Streams application from 0.10.0 to 0.11.0 does require a [broker upgrade] because a Kafka Strea brokers.
- There are couple of API changes, that are not backward compatible (cf. [Streams API changes in 0.11.0], [Stream more details). Thus, you need to update and recompile your code. Just swapping the Kafka Streams library jar f
- Upgrading from 0.10.0.x to 0.11.0.3 requires two rolling bounces with config `upgrade.from="0.10.0"` se upgrade is also possible.

- prepare your application instances for a rolling bounce and make sure that config `upgrade.from` is set t
- bounce each instance of your application once
- prepare your newly deployed 0.11.0.3 application instances for a second round of rolling bounces; make sur
- bounce each instance of your application once more to complete the upgrade

* Upgrading from 0.10.0.x to 0.11.0.0, 0.11.0.1, or 0.11.0.2 requires an offline upgrade (rolling bounce upgrade is

  - stop all old (0.10.0.x) application instances
  - update your code and swap old code and jar file with new code and new jar file
  - restart all new (0.11.0.0 , 0.11.0.1, or 0.11.0.2) application instances

Notable changes in 0.11.0.3

- New Kafka Streams configuration parameter `upgrade.from` added that allows rolling bounce upgrade from
- See the **Kafka Streams upgrade guide** for details about this new config.

Notable changes in 0.11.0.0

- Unclean leader election is now disabled by default. The new default favors durability over availability. Users wh config `unclean.leader.election.enable` to `true` .
- Producer configs `block.on.buffer.full` , `metadata.fetch.timeout.ms` and `timeout.ms` have
- The `offsets.topic.replication.factor` broker config is now enforced upon auto topic creation. Inte GROUP_COORDINATOR_NOT_AVAILABLE error until the cluster size meets this replication factor requirement.
- When compressing data with snappy, the producer and broker will use the compression scheme's default block compression ratio. There have been reports of data compressed with the smaller block size being 50% larger tl case, a producer with 5000 partitions will require an additional 315 MB of JVM heap.
- Similarly, when compressing data with gzip, the producer and broker will use 8 KB instead of 1 KB as the buffer
- The broker configuration `max.message.bytes` now applies to the total size of a batch of messages. Previo non-compressed messages individually. A message batch may consist of only a single message, so in most ca reduced by the overhead of the batch format. However, there are some subtle implications for message format previously the broker would ensure that at least one message is returned in each fetch request (regardless of tl applies to one message batch.
- GC log rotation is enabled by default, see KAFKA-3754 for details.
- Deprecated constructors of RecordMetadata, MetricName and Cluster classes have been removed.
- Added user headers support through a new Headers interface providing user headers read and write access.
- ProducerRecord and ConsumerRecord expose the new Headers API via `Headers headers()` method call.
- ExtendedSerializer and ExtendedDeserializer interfaces are introduced to support serialization and deserializati and deserializer are not the above classes.
- A new config, `group.initial.rebalance.delay.ms` , was introduced. This config specifies the time, in consumer rebalance. The rebalance will be further delayed by the value of `group.initial.rebalance.de` `max.poll.interval.ms` . The default value for this is 3 seconds. During development and testing it might time.

- `org.apache.kafka.common.Cluster#partitionsForTopic` , `partitionsForNode` and `availa` instead of `null` (which is considered a bad practice) in case the metadata for the required topic does not ex
- Streams API configuration parameters `timestamp.extractor` , `key.serde` , and `value.serde` were `default.timestamp.extractor` , `default.key.serde` , and `default.value.serde` , respective
- For offset commit failures in the Java consumer's `commitAsync` APIs, we no longer expose the underlying c `RetriableCommitFailedException` are passed to the commit callback. See KAFKA-5052 for more deta

New Protocol Versions

- KIP-107: FetchRequest v5 introduces a partition-level `log_start_offset` field.
- KIP-107: FetchResponse v5 introduces a partition-level `log_start_offset` field.
- KIP-82: ProduceRequest v3 introduces an array of `header` in the message protocol, containing `key` field a
- KIP-82: FetchResponse v5 introduces an array of `header` in the message protocol, containing `key` field an

Notes on Exactly Once Semantics

Kafka 0.11.0 includes support for idempotent and transactional capabilities in the producer. Idempotent delivery e topic partition during the lifetime of a single producer. Transactional delivery allows producers to send data to mul delivered, or none of them are. Together, these capabilities enable "exactly once semantics" in Kafka. More details a few specific notes on enabling them in an upgraded cluster. Note that enabling EoS is not required and there is n

1. Only the new Java producer and consumer support exactly once semantics.
2. These features depend crucially on the 0.11.0 message format. Attempting to use them on an older format w
3. Transaction state is stored in a new internal topic `__transaction_state` . This topic is not created until the consumer offsets topic, there are several settings to control the topic's configuration. For example, `tran` this topic. See the configuration section in the user guide for a full list of options.
4. For secure clusters, the transactional APIs require new ACLs which can be turned on with the `bin/kafka-a`
5. EoS in Kafka introduces new request APIs and modifies several existing ones. See KIP-98 for the full details

Notes on the new message format in 0.11.0

The 0.11.0 message format includes several major enhancements in order to support better delivery semantics fo tolerance (see KIP-101). Although the new format contains more information to make these improvements possib the number of messages per batch is more than 2, you can expect lower overall overhead. For smaller batches, ho results of our initial performance analysis of the new message format. You can also find more detail on the messa

One of the notable differences in the new message format is that even uncompressed messages are stored togetl configuration `max.message.bytes` , which limits the size of a single batch. First, if an older client produces me are individually smaller than `max.message.bytes` , the broker may still reject them after they are merged into a happen when the aggregate size of the individual messages is larger than `max.message.bytes` . There is a sir from the new format: if the fetch size is not set at least as large as `max.message.bytes` , the consumer may n messages are smaller than the configured fetch size. This behavior does not impact the Java client for 0.10.1.0 ar at least one message can be returned even if it exceeds the fetch size. To get around these problems, you should `max.message.bytes` , and 2) that the consumer's fetch size is set at least as large as `max.message.bytes`

Most of the discussion on the performance impact of [upgrading to the 0.10.0 message format] remains pertinent t secured with TLS since "zero-copy" transfer is already not possible in that case. In order to avoid the cost of down- upgraded to the latest 0.11.0 client. Significantly, since the old consumer has been deprecated in 0.11.0.0, it does new consumer to use the new message format without the cost of down-conversion. Note that 0.11.0 consumers so it is possible to upgrade the clients first before the brokers.

Upgrading from 0.8.x, 0.9.x, 0.10.0.x or 0.10.1.x to 0.10.2.0

0.10.2.0 has wire protocol changes. By following the recommended rolling upgrade plan below, you guarantee no changes in 0.10.2.0 before upgrading.

Starting with version 0.10.2, Java clients (producer and consumer) have acquired the ability to communicate with newer brokers. However, if your brokers are older than 0.10.0, you must upgrade all the brokers in the Kafka cluste 0.8.x and newer clients.

For a rolling upgrade:

1. Update server.properties file on all brokers and add the following properties:
    - inter.broker.protocol.version=CURRENT_KAFKA_VERSION (e.g. 0.8.2, 0.9.0, 0.10.0 or 0.10.1).
    - log.message.format.version=CURRENT_KAFKA_VERSION (See potential performance impact following the
2. Upgrade the brokers one at a time: shut down the broker, update the code, and restart it.
3. Once the entire cluster is upgraded, bump the protocol version by editing inter.broker.protocol.version and set
4. If your previous message format is 0.10.0, change log.message.format.version to 0.10.2 (this is a no-op as th your previous message format version is lower than 0.10.0, do not change log.message.format.version yet - t upgraded to 0.10.0.0 or later.
5. Restart the brokers one by one for the new protocol version to take effect.
6. If log.message.format.version is still lower than 0.10.0 at this point, wait until all consumers have been upgra 0.10.2 on each broker and restart them one by one.

Note: If you are willing to accept downtime, you can simply take all the brokers down, update the code and start al

Note: Bumping the protocol version and restarting can be done any time after the brokers were upgraded. It does

Upgrading a 0.10.1 Kafka Streams Application

- Upgrading your Streams application from 0.10.1 to 0.10.2 does not require a broker upgrade. A Kafka Streams not possible to connect to 0.10.0 brokers though).
- You need to recompile your code. Just swapping the Kafka Streams library jar file will not work and will break y
- If you use a custom (i.e., user implemented) timestamp extractor, you will need to update this code, because th
- If you register custom metrics, you will need to update this code, because the `StreamsMetric` interface wa
- See Streams API changes in 0.10.2 for more details.

Upgrading a 0.10.0 Kafka Streams Application

- Upgrading your Streams application from 0.10.0 to 0.10.2 does require a broker upgrade because a Kafka Strea
- There are couple of API changes, that are not backward compatible (cf. Streams API changes in 0.10.2 for mor swapping the Kafka Streams library jar file will not work and will break your application.
- Upgrading from 0.10.0.x to 0.10.2.2 requires two rolling bounces with config `upgrade.from="0.10.0"` se upgrade is also possible.
    - prepare your application instances for a rolling bounce and make sure that config `upgrade.from` is set t
    - bounce each instance of your application once
    - prepare your newly deployed 0.10.2.2 application instances for a second round of rolling bounces; make sur
    - bounce each instance of your application once more to complete the upgrade
- Upgrading from 0.10.0.x to 0.10.2.0 or 0.10.2.1 requires an offline upgrade (rolling bounce upgrade is not suppo
    - stop all old (0.10.0.x) application instances
    - update your code and swap old code and jar file with new code and new jar file
    - restart all new (0.10.2.0 or 0.10.2.1) application instances

Notable changes in 0.10.2.2

- New configuration parameter `upgrade.from` added that allows rolling bounce upgrade from version 0.10.0

Notable changes in 0.10.2.1

- The default values for two configurations of the StreamsConfig class were changed to improve the resiliency of `retries` default value was changed from 0 to 10. The internal Kafka Streams consumer `max.poll.inte` `Integer.MAX_VALUE` .

Notable changes in 0.10.2.0

- The Java clients (producer and consumer) have acquired the ability to communicate with older brokers. Versio that some features are not available or are limited when older brokers are used.
- Several methods on the Java consumer may now throw `InterruptException` if the calling thread is interr in-depth explanation of this change.
- Java consumer now shuts down gracefully. By default, the consumer waits up to 30 seconds to complete pend `KafkaConsumer` to control the maximum wait time.
- Multiple regular expressions separated by commas can be passed to MirrorMaker with the new Java consume with MirrorMaker when used the old Scala consumer.
- Upgrading your Streams application from 0.10.1 to 0.10.2 does not require a broker upgrade. A Kafka Streams not possible to connect to 0.10.0 brokers though).
- The Zookeeper dependency was removed from the Streams API. The Streams API now uses the Kafka protoco This eliminates the need for privileges to access Zookeeper directly and "StreamsConfig.ZOOKEEPER_CONFIG" cluster is secured, Streams apps must have the required security privileges to create new topics.

- Several new fields including "security.protocol", "connections.max.idle.ms", "retry.backoff.ms", "reconnect.backo class. User should pay attention to the default values and set these if needed. For more details please refer to 3

New Protocol Versions

- KIP-88: OffsetFetchRequest v2 supports retrieval of offsets for all topics if the `topics` array is set to `null`
- KIP-88: OffsetFetchResponse v2 introduces a top-level `error_code` field.
- KIP-103: UpdateMetadataRequest v3 introduces a `listener_name` field to the elements of the `end_poin`
- KIP-108: CreateTopicsRequest v1 introduces a `validate_only` field.
- KIP-108: CreateTopicsResponse v1 introduces an `error_message` field to the elements of the `topic_er`

Upgrading from 0.8.x, 0.9.x or 0.10.0.X to 0.10.1.0

0.10.1.0 has wire protocol changes. By following the recommended rolling upgrade plan below, you guarantee no breaking changes in 0.10.1.0 before upgrade.
Note: Because new protocols are introduced, it is important to upgrade your Kafka clusters before upgrading your while 0.10.1.x brokers also support older clients).

For a rolling upgrade:

1. Update server.properties file on all brokers and add the following properties:
   - inter.broker.protocol.version=CURRENT_KAFKA_VERSION (e.g. 0.8.2.0, 0.9.0.0 or 0.10.0.0).
   - log.message.format.version=CURRENT_KAFKA_VERSION (See potential performance impact following the
2. Upgrade the brokers one at a time: shut down the broker, update the code, and restart it.
3. Once the entire cluster is upgraded, bump the protocol version by editing inter.broker.protocol.version and set
4. If your previous message format is 0.10.0, change log.message.format.version to 0.10.1 (this is a no-op as th previous message format version is lower than 0.10.0, do not change log.message.format.version yet - this pa upgraded to 0.10.0.0 or later.
5. Restart the brokers one by one for the new protocol version to take effect.
6. If log.message.format.version is still lower than 0.10.0 at this point, wait until all consumers have been upgra 0.10.1 on each broker and restart them one by one.

Note: If you are willing to accept downtime, you can simply take all the brokers down, update the code and start al

Note: Bumping the protocol version and restarting can be done any time after the brokers were upgraded. It does n

Potential breaking changes in 0.10.1.0

- The log retention time is no longer based on last modified time of the log segments. Instead it will be based on
- The log rolling time is no longer depending on log segment create time. Instead it is now based on the timestar message in the segment is T, the log will be rolled out when a new message has a timestamp greater than or ec
- The open file handlers of 0.10.0 will increase by ~33% because of the addition of time index files for each segm

- The time index and offset index share the same index size configuration. Since each time index entry is 1.5x th log.index.size.max.bytes to avoid potential frequent log rolling.
- Due to the increased number of index files, on some brokers with large amount the log segments (e.g. >15K), th Based on our experiment, setting the num.recovery.threads.per.data.dir to one may reduce the log loading time.

Upgrading a 0.10.0 Kafka Streams Application

- Upgrading your Streams application from 0.10.0 to 0.10.1 does require a broker upgrade because a Kafka Strea
- There are couple of API changes, that are not backward compatible (cf. Streams API changes in 0.10.1 for mor swapping the Kafka Streams library jar file will not work and will break your application.
- Upgrading from 0.10.0.x to 0.10.1.2 requires two rolling bounces with config `upgrade.from="0.10.0"` se upgrade is also possible.
    - prepare your application instances for a rolling bounce and make sure that config `upgrade.from` is set t
    - bounce each instance of your application once
    - prepare your newly deployed 0.10.1.2 application instances for a second round of rolling bounces; make sur
    - bounce each instance of your application once more to complete the upgrade
- Upgrading from 0.10.0.x to 0.10.1.0 or 0.10.1.1 requires an offline upgrade (rolling bounce upgrade is not suppo
    - stop all old (0.10.0.x) application instances
    - update your code and swap old code and jar file with new code and new jar file
    - restart all new (0.10.1.0 or 0.10.1.1) application instances

Notable changes in 0.10.1.0

- The new Java consumer is no longer in beta and we recommend it for all new development. The old Scala cons release and will be removed in a future major release.
- The `--new-consumer` / `--new.consumer` switch is no longer required to use tools like MirrorMaker and to pass a Kafka broker to connect to instead of the ZooKeeper ensemble. In addition, usage of the Console Cor removed in a future major release.
- Kafka clusters can now be uniquely identified by a cluster id. It will be automatically generated when a broker is kafka.server:type=KafkaServer,name=ClusterId metric and it is part of the Metadata response. Serializers, clien implementing the ClusterResourceListener interface.
- The BrokerState "RunningAsController" (value 4) has been removed. Due to a bug, a broker would only be in this the removal should be minimal. The recommended way to detect if a given broker is the controller is via the kaf metric.
- The new Java Consumer now allows users to search offsets by timestamp on partitions.
- The new Java Consumer now supports heartbeating from a background thread. There is a new configuration `r` between poll invocations before the consumer will proactively leave the group (5 minutes by default). The value larger than `max.poll.interval.ms` because this is the maximum time that a JoinGroup request can bloc changed its default value to just above 5 minutes. Finally, the default value of `session.timeout.ms` has b `max.poll.records` has been changed to 500.

- When using an Authorizer and a user doesn't have **Describe** authorization on a topic, the broker will no longer re leaks topic names. Instead, the UNKNOWN_TOPIC_OR_PARTITION error code will be returned. This may cause consumer since Kafka clients will typically retry automatically on unknown topic errors. You should consult the
- Fetch responses have a size limit by default (50 MB for consumers and 10 MB for replication). The existing per Note that neither of these limits is an absolute maximum as explained in the next point.
- Consumers and replicas can make progress if a message larger than the response/partition size limit is found. partition of the fetch is larger than either or both limits, the message will still be returned.
- Overloaded constructors were added to `kafka.api.FetchRequest` and `kafka.javaapi.FetchReque` order is significant in v3). The previously existing constructors were deprecated and the partitions are shuffled l

New Protocol Versions

- ListOffsetRequest v1 supports accurate offset search based on timestamps.
- MetadataResponse v2 introduces a new field: "cluster_id".
- FetchRequest v3 supports limiting the response size (in addition to the existing per partition limit), it returns me order of partitions in the request is now significant.
- JoinGroup v1 introduces a new field: "rebalance_timeout".

Upgrading from 0.8.x or 0.9.x to 0.10.0.0

0.10.0.0 has potential breaking changes (please review before upgrading) and possible performance impact follov plan below, you guarantee no downtime and no performance impact during and following the upgrade.
Note: Because new protocols are introduced, it is important to upgrade your Kafka clusters before upgrading your

Notes to clients with version 0.9.0.0: Due to a bug introduced in 0.9.0.0, clients that depend on ZooKeeper (old Sc consumer) will not work with 0.10.0.x brokers. Therefore, 0.9.0.0 clients should be upgraded to 0.9.0.1 **before** brok or 0.9.0.1 clients.

For a rolling upgrade:

1. Update server.properties file on all brokers and add the following properties:
    - inter.broker.protocol.version=CURRENT_KAFKA_VERSION (e.g. 0.8.2 or 0.9.0.0).
    - log.message.format.version=CURRENT_KAFKA_VERSION (See potential performance impact following the
2. Upgrade the brokers. This can be done a broker at a time by simply bringing it down, updating the code, and re
3. Once the entire cluster is upgraded, bump the protocol version by editing inter.broker.protocol.version and set log.message.format.version yet - this parameter should only change once all consumers have been upgraded
4. Restart the brokers one by one for the new protocol version to take effect.
5. Once all consumers have been upgraded to 0.10.0, change log.message.format.version to 0.10.0 on each bro

Note: If you are willing to accept downtime, you can simply take all the brokers down, update the code and start al

Note: Bumping the protocol version and restarting can be done any time after the brokers were upgraded. It does r

Potential performance impact following upgrade to 0.10.0.0

The message format in 0.10.0 includes a new timestamp field and uses relative offsets for compressed messages log.message.format.version in the server.properties file. The default on-disk message format is 0.10.0. If a consur message formats before 0.10.0. In this case, the broker is able to convert messages from the 0.10.0 format to an older version. However, the broker can't use zero-copy transfer in this case. Reports from the Kafka community on 20% before to 100% after an upgrade, which forced an immediate upgrade of all clients to bring performance back are upgraded to 0.10.0.0, one can set log.message.format.version to 0.8.2 or 0.9.0 when upgrading the broker to 0 send the data to the old consumers. Once consumers are upgraded, one can change the message format to 0.10.(new timestamp and improved compression. The conversion is supported to ensure compatibility and can be usefu but is impractical to support all consumer traffic on even an overprovisioned cluster. Therefore, it is critical to avoi been upgraded but the majority of clients have not.

For clients that are upgraded to 0.10.0.0, there is no performance impact.

Note: By setting the message format version, one certifies that all existing messages are on or below that messag break. In particular, after the message format is set to 0.10.0, one should not change it back to an earlier format as

Note: Due to the additional timestamp introduced in each message, producers sending small messages may see a overhead. Likewise, replication now transmits an additional 8 bytes per message. If you're running close to the net network cards and see failures and performance issues due to the overload.

Note: If you have enabled compression on producers, you may notice reduced producer throughput and/or lower c compressed messages, 0.10.0 brokers avoid recompressing the messages, which in general reduces the latency a reduce the batching size on the producer, which could lead to worse throughput. If this happens, users can tune lir addition, the producer buffer used for compressing messages with snappy is smaller than the one used by the bro for the messages on disk. We intend to make this configurable in a future Kafka release.

Potential breaking changes in 0.10.0.0

- Starting from Kafka 0.10.0.0, the message format version in Kafka is represented as the Kafka version. For exa supported by Kafka 0.9.0.
- Message format 0.10.0 has been introduced and it is used by default. It includes a timestamp field in the mess
- ProduceRequest/Response v2 has been introduced and it is used by default to support message format 0.10.0
- FetchRequest/Response v2 has been introduced and it is used by default to support message format 0.10.0
- MessageFormatter interface was changed from `def writeTo(key: Array[Byte], value: Array[By` `writeTo(consumerRecord: ConsumerRecord[Array[Byte], Array[Byte]], output: PrintSt`
- MessageReader interface was changed from `def readMessage(): KeyedMessage[Array[Byte], Ar` `ProducerRecord[Array[Byte], Array[Byte]]`
- MessageFormatter's package was changed from `kafka.tools` to `kafka.common`
- MessageReader's package was changed from `kafka.tools` to `kafka.common`
- MirrorMakerMessageHandler no longer exposes the `handle(record: MessageAndMetadata[Array[By`
- The 0.7 KafkaMigrationTool is no longer packaged with Kafka. If you need to migrate from 0.7 to 0.10.0, please process to upgrade from 0.8 to 0.10.0.

- The new consumer has standardized its APIs to accept `java.util.Collection` as the sequence type for work with the 0.10.0 client library.
- LZ4-compressed message handling was changed to use an interoperable framing specification (LZ4f v1.5.1). T to Message format 0.10.0 and later. Clients that Produce/Fetch LZ4-compressed messages using v0/v1 (Mess implementation. Clients that use Produce/Fetch protocols v2 or later should use interoperable LZ4f framing. A

Notable changes in 0.10.0.0

- Starting from Kafka 0.10.0.0, a new client library named **Kafka Streams** is available for stream processing on d 0.10.x and upward versioned brokers due to message format changes mentioned above. For more information
- The default value of the configuration parameter `receive.buffer.bytes` is now 64K for the new consum
- The new consumer now exposes the configuration parameter `exclude.internal.topics` to restrict inter being included in regular expression subscriptions. By default, it is enabled.
- The old Scala producer has been deprecated. Users should migrate their code to the Java producer included in
- The new consumer API has been marked stable.

Upgrading from 0.8.0, 0.8.1.X, or 0.8.2.X to 0.9.0.0

0.9.0.0 has potential breaking changes (please review before upgrading) and an inter-broker protocol change from may not be compatible with older versions. It is important that you upgrade your Kafka cluster before upgrading yo should be upgraded first as well.

For a rolling upgrade:

1. Update server.properties file on all brokers and add the following property: inter.broker.protocol.version=0.8.2
2. Upgrade the brokers. This can be done a broker at a time by simply bringing it down, updating the code, and re
3. Once the entire cluster is upgraded, bump the protocol version by editing inter.broker.protocol.version and set
4. Restart the brokers one by one for the new protocol version to take effect

Note: If you are willing to accept downtime, you can simply take all the brokers down, update the code and start al

Note: Bumping the protocol version and restarting can be done any time after the brokers were upgraded. It does r

Potential breaking changes in 0.9.0.0

- Java 1.6 is no longer supported.
- Scala 2.9 is no longer supported.
- Broker IDs above 1000 are now reserved by default to automatically assigned broker IDs. If your cluster has exi reserved.broker.max.id broker configuration property accordingly.
- Configuration parameter replica.lag.max.messages was removed. Partition leaders will no longer consider the r sync.
- Configuration parameter replica.lag.time.max.ms now refers not just to the time passed since last fetch reques Replicas that are still fetching messages from leaders but did not catch up to the latest messages in replica.lag

- Compacted topics no longer accept messages without key and an exception is thrown by the producer if this is compaction thread to subsequently complain and quit (and stop compacting all compacted topics).
- MirrorMaker no longer supports multiple target clusters. As a result it will only accept a single --consumer.confi least one MirrorMaker instance per source cluster, each with its own consumer configuration.
- Tools packaged under *org.apache.kafka.clients.tools.** have been moved to *org.apache.kafka.tools.**. All includ importing these classes will be affected.
- The default Kafka JVM performance options (KAFKA_JVM_PERFORMANCE_OPTS) have been changed in kafk
- The kafka-topics.sh script (kafka.admin.TopicCommand) now exits with non-zero exit code on failure.
- The kafka-topics.sh script (kafka.admin.TopicCommand) will now print a warning when topic names risk metric the case of an actual collision.
- The kafka-console-producer.sh script (kafka.tools.ConsoleProducer) will use the Java producer instead of the o producer' to use the old producer.
- By default, all command line tools will print all logging messages to stderr instead of stdout.

Notable changes in 0.9.0.1

- The new broker id generation feature can be disabled by setting broker.id.generation.enable to false.
- Configuration parameter log.cleaner.enable is now true by default. This means topics with a cleanup.policy=cor be allocated to the cleaner process via log.cleaner.dedupe.buffer.size. You may want to review log.cleaner.dedu on your usage of compacted topics.
- Default value of configuration parameter fetch.min.bytes for the new consumer is now 1 by default.

Deprecations in 0.9.0.0

- Altering topic configuration from the kafka-topics.sh script (kafka.admin.TopicCommand) has been deprecatec (kafka.admin.ConfigCommand) for this functionality.
- The kafka-consumer-offset-checker.sh (kafka.tools.ConsumerOffsetChecker) has been deprecated. Going forw (kafka.admin.ConsumerGroupCommand) for this functionality.
- The kafka.tools.ProducerPerformance class has been deprecated. Going forward, please use org.apache.kafka perf-test.sh will also be changed to use the new class).
- The producer config block.on.buffer.full has been deprecated and will be removed in future release. Currently its no longer throw BufferExhaustedException but instead will use max.block.ms value to block, after which it will t true explicitly, it will set the max.block.ms to Long.MAX_VALUE and metadata.fetch.timeout.ms will not be hon

Upgrading from 0.8.1 to 0.8.2

0.8.2 is fully compatible with 0.8.1. The upgrade can be done one broker at a time by simply bringing it down, upda

Upgrading from 0.8.0 to 0.8.1

0.8.1 is fully compatible with 0.8. The upgrade can be done one broker at a time by simply bringing it down, updati

Upgrading from 0.7

Release 0.7 is incompatible with newer releases. Major changes were made to the API, ZooKeeper data structures was missing in 0.7). The upgrade from 0.7 to later versions requires a special tool for migration. This migration ca

2. APIS

Kafka includes five core apis:

1. The Producer API allows applications to send streams of data to topics in the Kafka cluster.
2. The Consumer API allows applications to read streams of data from topics in the Kafka cluster.
3. The Streams API allows transforming streams of data from input topics to output topics.
4. The Connect API allows implementing connectors that continually pull from some source system or applicati application.
5. The Admin API allows managing and inspecting topics, brokers, and other Kafka objects.

Kafka exposes all its functionality over a language independent protocol which has clients available in many progr as part of the main Kafka project, the others are available as independent open source projects. A list of non-Java

2.1 Producer API

The Producer API allows applications to send streams of data to topics in the Kafka cluster.

Examples showing how to use the producer are given in the javadocs.

To use the producer, you can use the following maven dependency:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.4.0</version>
</dependency>

```

2.2 Consumer API

The Consumer API allows applications to read streams of data from topics in the Kafka cluster.

Examples showing how to use the consumer are given in the javadocs.

To use the consumer, you can use the following maven dependency:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.4.0</version>
</dependency>

```

2.3 Streams API

The Streams API allows transforming streams of data from input topics to output topics.

Examples showing how to use this library are given in the javadocs

Additional documentation on using the Streams API is available here.

To use Kafka Streams you can use the following maven dependency:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams</artifactId>
    <version>2.4.0</version>
</dependency>

```

When using Scala you may optionally include the `kafka-streams-scala` library. Additional documentation on developer guide.

To use Kafka Streams DSL for Scala for Scala 2.12 you can use the following maven dependency:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams-scala_2.12</artifactId>
    <version>2.4.0</version>
</dependency>

```

2.4 Connect API

The Connect API allows implementing connectors that continually pull from some source data system into Kafka

Many users of Connect won't need to use this API directly, though, they can use pre-built connectors without needi available here.

Those who want to implement custom connectors can see the javadoc.

2.5 Admin API

The Admin API supports managing and inspecting topics, brokers, acls, and other Kafka objects.

To use the Admin API, add the following Maven dependency:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.4.0</version>
</dependency>

```

For more information about the Admin APIs, see the javadoc.

3. CONFIGURATION

Kafka uses key-value pairs in the property file format for configuration. These values can be supplied either from a

3.1 Broker Configs

The essential configurations are the following:

- `broker.id`
- `log.dirs`
- `zookeeper.connect`

Topic-level configurations and defaults are discussed in more detail below.

zookeeper.connect: Specifies the ZooKeeper connection string in the form `hostname:port` where host a connecting through other ZooKeeper nodes when that ZooKeeper machine is down you can also specify mul `hostname1:port1,hostname2:port2,hostname3:port3` . The server can also have a ZooKeeper ch data under some path in the global ZooKeeper namespace. For example to give a chroot path of `/chroot/` `hostname1:port1,hostname2:port2,hostname3:port3/chroot/path` .

Type: string — **Default**: — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

advertised.host.name: DEPRECATED: only used when `advertised.listeners` or `listeners` are nc publish to ZooKeeper for clients to use. In IaaS environments, this may need to be different from the interfac `host.name` if configured. Otherwise it will use the value returned from java.net.InetAddress.getCanonicalI

Type: string — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

advertised.listeners: Listeners to publish to ZooKeeper for clients to use, if different than the `listeners` from the interface to which the broker binds. If this is not set, the value for `listeners` will be used. Unlike address.

Type: string — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: per-broker

---

advertised.port: DEPRECATED: only used when `advertised.listeners` or `listeners` are not set. U ZooKeeper for clients to use. In IaaS environments, this may need to be different from the port to which the b broker binds to.

Type: int — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

auto.create.topics.enable: Enable auto creation of topic on the server

Type: boolean — **Default**: true — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

auto.leader.rebalance.enable: Enables auto leader balancing. A background thread checks the distribution of `leader.imbalance.check.interval.seconds`. If the leader imbalance exceeds `leader.imbalance.per.broker.perc triggered.

Type: boolean — **Default**: true — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

background.threads: The number of threads to use for various background processing tasks

Type: int — **Default**: 10 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: cluster-wide

---

broker.id: The broker id for this server. If unset, a unique broker id will be generated.To avoid conflicts betwee generated broker ids start from reserved.broker.max.id + 1.

Type: int — **Default**: -1 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

compression.type: Specify the final compression type for a given topic. This configuration accepts the stand accepts 'uncompressed' which is equivalent to no compression; and 'producer' which means retain the origin

Type: string — **Default**: producer — **Valid Values**: — **Importance**: high — **Update Mode**: cluster-wide

---

control.plane.listener.name: Name of listener used for communication between controller and brokers. Brok listeners list, to listen for connections from the controller. For example, if a broker's config is : listeners = INTI CONTROLLER://192.1.1.8:9094 listener.security.protocol.map = INTERNAL:PLAINTEXT, EXTERNAL:SSL, CON startup, the broker will start listening on "192.1.1.8:9094" with security protocol "SSL". On controller side, whe will use the control.plane.listener.name to find the endpoint, which it will use to establish connection to the b zookeeper are : "endpoints" : ["INTERNAL://broker1.example.com:9092","EXTERNAL://broker1.example.com: controller's config is : listener.security.protocol.map = INTERNAL:PLAINTEXT, EXTERNAL:SSL, CONTROLLER: will use "broker1.example.com:9094" with security protocol "SSL" to connect to the broker. If not explicitly co endpoints for controller connections.

Type: string — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

delete.topic.enable: Enables delete topic. Delete topic through the admin tool will have no effect if this confi

Type: boolean — **Default**: true — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

host.name: DEPRECATED: only used when `listeners` is not set. Use `listeners` instead. hostname c set, it will bind to all interfaces

Type: string — **Default**: "" — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

leader.imbalance.check.interval.seconds: The frequency with which the partition rebalance check is triggere

Type: long — **Default**: 300 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

leader.imbalance.per.broker.percentage: The ratio of leader imbalance allowed per broker. The controller wo The value is specified in percentage.

Type: int — **Default**: 10 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

listeners: Listener List - Comma-separated list of URIs we will listen on and the listener names. If the listener
must also be set. Specify hostname as 0.0.0.0 to bind to all interfaces. Leave hostname empty to bind to def
PLAINTEXT://myhost:9092,SSL://:9091 CLIENT://0.0.0.0:9092,REPLICATION://localhost:9093

Type: string — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: per-broker

---

log.dir: The directory in which the log data is kept (supplemental for log.dirs property)

Type: string — **Default**: /tmp/kafka-logs — **Valid Values**: — **Importance**: high — **Update Mode**: read-onl

---

log.dirs: The directories in which the log data is kept. If not set, the value in log.dir is used

Type: string — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

log.flush.interval.messages: The number of messages accumulated on a log partition before messages are

Type: long — **Default**: 9223372036854775807 — **Valid Values**: [1,...] — **Importance**: high — **Update Mo**

---

log.flush.interval.ms: The maximum time in ms that a message in any topic is kept in memory before flushe
used

Type: long — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: cluster-wide

---

log.flush.offset.checkpoint.interval.ms: The frequency with which we update the persistent record of the las

Type: int — **Default**: 60000 — **Valid Values**: [0,...] — **Importance**: high — **Update Mode**: read-only

---

log.flush.scheduler.interval.ms: The frequency in ms that the log flusher checks whether any log needs to be

Type: long — **Default**: 9223372036854775807 — **Valid Values**: — **Importance**: high — **Update Mode**: re

---

log.flush.start.offset.checkpoint.interval.ms: The frequency with which we update the persistent record of lo

Type: int — **Default**: 60000 — **Valid Values**: [0,...] — **Importance**: high — **Update Mode**: read-only

---

log.retention.bytes: The maximum size of the log before deleting it

Type: long — **Default**: -1 — **Valid Values**: — **Importance**: high — **Update Mode**: cluster-wide

---

log.retention.hours: The number of hours to keep a log file before deleting it (in hours), tertiary to log.retentic

Type: int — **Default**: 168 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

log.retention.minutes: The number of minutes to keep a log file before deleting it (in minutes), secondary to l
is used

Type: int — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

log.retention.ms: The number of milliseconds to keep a log file before deleting it (in milliseconds), If not set,
is applied.

Type: long — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: cluster-wide

---

log.roll.hours: The maximum time before a new log segment is rolled out (in hours), secondary to log.roll.ms

Type: int — **Default**: 168 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

log.roll.jitter.hours: The maximum jitter to subtract from logRollTimeMillis (in hours), secondary to log.roll.jit

Type: int — **Default**: 0 — **Valid Values**: [0,...] — **Importance**: high — **Update Mode**: read-only

---

log.roll.jitter.ms: The maximum jitter to subtract from logRollTimeMillis (in milliseconds). If not set, the value

Type: long — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: cluster-wide

---

log.roll.ms: The maximum time before a new log segment is rolled out (in milliseconds). If not set, the value

Type: long — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: cluster-wide

---

log.segment.bytes: The maximum size of a single log file

Type: int — **Default**: 1073741824 — **Valid Values**: [14,...] — **Importance**: high — **Update Mode**: cluster-v

---

log.segment.delete.delay.ms: The amount of time to wait before deleting a file from the filesystem

Type: long — **Default**: 60000 — **Valid Values**: [0,...] — **Importance**: high — **Update Mode**: cluster-wide

---

message.max.bytes: The largest record batch size allowed by Kafka. If this is increased and there are consu
increased so that the they can fetch record batches this large. In the latest message format version, records
message format versions, uncompressed records are not grouped into batches and this limit only applies to
level `max.message.bytes` config.

Type: int — **Default**: 1000012 — **Valid Values**: [0,...] — **Importance**: high — **Update Mode**: cluster-wide

---

min.insync.replicas: When a producer sets acks to "all" (or "-1"), min.insync.replicas specifies the minimum n
be considered successful. If this minimum cannot be met, then the producer will raise an exception (either N
When used together, min.insync.replicas and acks allow you to enforce greater durability guarantees. A typic
set min.insync.replicas to 2, and produce with acks of "all". This will ensure that the producer raises an excep

Type: int — **Default**: 1 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: cluster-wide

---

num.io.threads: The number of threads that the server uses for processing requests, which may include disk

Type: int — **Default**: 8 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: cluster-wide

---

num.network.threads: The number of threads that the server uses for receiving requests from the network an

Type: int — **Default**: 3 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: cluster-wide

---

num.recovery.threads.per.data.dir: The number of threads per data directory to be used for log recovery at st

Type: int — **Default**: 1 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: cluster-wide

---

num.replica.alter.log.dirs.threads: The number of threads that can move replicas between log directories, wh

Type: int — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

num.replica.fetchers: Number of fetcher threads used to replicate messages from a source broker. Increasin follower broker.

Type: int — **Default**: 1 — **Valid Values**: — **Importance**: high — **Update Mode**: cluster-wide

---

offset.metadata.max.bytes: The maximum size for a metadata entry associated with an offset commit

Type: int — **Default**: 4096 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

offsets.commit.required.acks: The required acks before the commit can be accepted. In general, the default

Type: short — **Default**: -1 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

offsets.commit.timeout.ms: Offset commit will be delayed until all replicas for the offsets topic receive the c request timeout.

Type: int — **Default**: 5000 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

offsets.load.buffer.size: Batch size for reading from the offsets segments when loading offsets into the cach

Type: int — **Default**: 5242880 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

offsets.retention.check.interval.ms: Frequency at which to check for stale offsets

Type: long — **Default**: 600000 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

offsets.retention.minutes: After a consumer group loses all its consumers (i.e. becomes empty) its offsets v standalone consumers (using manual assignment), offsets will be expired after the time of last commit plus

Type: int — **Default**: 10080 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

offsets.topic.compression.codec: Compression codec for the offsets topic - compression may be used to ac

Type: int — **Default**: 0 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

offsets.topic.num.partitions: The number of partitions for the offset commit topic (should not change after c

Type: int — **Default**: 50 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

offsets.topic.replication.factor: The replication factor for the offsets topic (set higher to ensure availability). replication factor requirement.

Type: short — **Default**: 3 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

offsets.topic.segment.bytes: The offsets topic segment bytes should be kept relatively small in order to facil

Type: int — **Default**: 104857600 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

port: DEPRECATED: only used when `listeners` is not set. Use `listeners` instead. the port to listen a

Type: int — **Default**: 9092 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

queued.max.requests: The number of queued requests allowed for data-plane, before blocking the network t

Type: int — **Default**: 500 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

quota.consumer.default: DEPRECATED: Used only when dynamic default quotas are not configured for or in : group will get throttled if it fetches more bytes than this value per-second

Type: long — **Default**: 9223372036854775807 — **Valid Values**: [1,...] — **Importance**: high — **Update Mo

---

quota.producer.default: DEPRECATED: Used only when dynamic default quotas are not configured for , or in : if it produces more bytes than this value per-second

Type: long — **Default**: 9223372036854775807 — **Valid Values**: [1,...] — **Importance**: high — **Update Mo

---

replica.fetch.min.bytes: Minimum bytes expected for each fetch response. If not enough bytes, wait up to re

Type: int — **Default**: 1 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

replica.fetch.wait.max.ms: max wait time for each fetcher request issued by follower replicas. This value sho prevent frequent shrinking of ISR for low throughput topics

Type: int — **Default**: 500 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

replica.high.watermark.checkpoint.interval.ms: The frequency with which the high watermark is saved out t

Type: long — **Default**: 5000 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

replica.lag.time.max.ms: If a follower hasn't sent any fetch requests or hasn't consumed up to the leaders lo follower from isr

Type: long — **Default**: 10000 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

replica.socket.receive.buffer.bytes: The socket receive buffer for network requests

Type: int — **Default**: 65536 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

replica.socket.timeout.ms: The socket timeout for network requests. Its value should be at least replica.fetc

Type: int — **Default**: 30000 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

request.timeout.ms: The configuration controls the maximum amount of time the client will wait for the resp timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.

Type: int — **Default**: 30000 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

socket.receive.buffer.bytes: The SO_RCVBUF buffer of the socket server sockets. If the value is -1, the OS de

Type: int — **Default**: 102400 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

socket.request.max.bytes: The maximum number of bytes in a socket request

Type: int — **Default**: 104857600 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

socket.send.buffer.bytes: The SO_SNDBUF buffer of the socket server sockets. If the value is -1, the OS defa

Type: int — **Default**: 102400 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

transaction.max.timeout.ms: The maximum allowed timeout for transactions. If a client's requested transact InitProducerIdRequest. This prevents a client from too large of a timeout, which can stall consumers reading

Type: int — **Default**: 900000 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

transaction.state.log.load.buffer.size: Batch size for reading from the transaction log segments when loadin overridden if records are too large).

Type: int — **Default**: 5242880 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

transaction.state.log.min.isr: Overridden min.insync.replicas config for the transaction topic.

Type: int — **Default**: 2 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

transaction.state.log.num.partitions: The number of partitions for the transaction topic (should not change a

Type: int — **Default**: 50 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

transaction.state.log.replication.factor: The replication factor for the transaction topic (set higher to ensure meets this replication factor requirement.

Type: short — **Default**: 3 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

transaction.state.log.segment.bytes: The transaction topic segment bytes should be kept relatively small in

Type: int — **Default**: 104857600 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

transactional.id.expiration.ms: The time in ms that the transaction coordinator will wait without receiving an expiring its transactional id. This setting also influences producer id expiration - producer ids are expired onc id. Note that producer ids may expire sooner if the last write from the producer id is deleted due to the topic's

Type: int — **Default**: 604800000 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

unclean.leader.election.enable: Indicates whether to enable replicas not in the ISR set to be elected as leade

Type: boolean — **Default**: false — **Valid Values**: — **Importance**: high — **Update Mode**: cluster-wide

---

zookeeper.connection.timeout.ms: The max time that the client waits to establish a connection to zookeepe

Type: int — **Default**: null — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

zookeeper.max.in.flight.requests: The maximum number of unacknowledged requests the client will send to

Type: int — **Default**: 10 — **Valid Values**: [1,...] — **Importance**: high — **Update Mode**: read-only

---

zookeeper.session.timeout.ms: Zookeeper session timeout

Type: int — **Default**: 6000 — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

zookeeper.set.acl: Set client to use secure ACLs

Type: boolean — **Default**: false — **Valid Values**: — **Importance**: high — **Update Mode**: read-only

---

broker.id.generation.enable: Enable automatic broker id generation on the server. When enabled the value cc

Type: boolean — **Default**: true — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

broker.rack: Rack of the broker. This will be used in rack aware replication assignment for fault tolerance. Exa

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

connections.max.idle.ms: Idle connections timeout: the server socket processor threads close the connectic

Type: long — **Default**: 600000 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

connections.max.reauth.ms: When explicitly set to a positive number (the default is 0, not a positive number communicated to v2.2.0 or later clients when they authenticate. The broker will disconnect any such connect is then subsequently used for any purpose other than re-authentication. Configuration names can optionally l lower-case. For example, listener.name.sasl_ssl.oauthbearer.connections.max.reauth.ms=3600000

Type: long — **Default**: 0 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

controlled.shutdown.enable: Enable controlled shutdown of the server

Type: boolean — **Default**: true — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

controlled.shutdown.max.retries: Controlled shutdown can fail for multiple reasons. This determines the nur

Type: int — **Default**: 3 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

controlled.shutdown.retry.backoff.ms: Before each retry, the system needs time to recover from the state th This config determines the amount of time to wait before retrying.

Type: long — **Default**: 5000 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

controller.socket.timeout.ms: The socket timeout for controller-to-broker channels

Type: int — **Default**: 30000 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

default.replication.factor: default replication factors for automatically created topics

Type: int — **Default**: 1 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

delegation.token.expiry.time.ms: The token validity time in miliseconds before the token needs to be renewe

Type: long — **Default**: 86400000 — **Valid Values**: [1,...] — **Importance**: medium — **Update Mode**: read-o

---

delegation.token.master.key: Master/secret key to generate and verify delegation tokens. Same key must be empty string, brokers will disable the delegation token support.

Type: password — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

delegation.token.max.lifetime.ms: The token has a maximum lifetime beyond which it cannot be renewed ar

Type: long — **Default**: 604800000 — **Valid Values**: [1,...] — **Importance**: medium — **Update Mode**: read-

---

delete.records.purgatory.purge.interval.requests: The purge interval (in number of requests) of the delete re

Type: int — **Default**: 1 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

fetch.purgatory.purge.interval.requests: The purge interval (in number of requests) of the fetch request purg

Type: int — **Default**: 1000 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

group.initial.rebalance.delay.ms: The amount of time the group coordinator will wait for more consumers to delay means potentially fewer rebalances, but increases the time until processing begins.

Type: int — **Default**: 3000 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

group.max.session.timeout.ms: The maximum allowed session timeout for registered consumers. Longer ti between heartbeats at the cost of a longer time to detect failures.

Type: int — **Default**: 1800000 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

group.max.size: The maximum number of consumers that a single consumer group can accommodate.

Type: int — **Default**: 2147483647 — **Valid Values**: [1,...] — **Importance**: medium — **Update Mode**: read-c

---

group.min.session.timeout.ms: The minimum allowed session timeout for registered consumers. Shorter tir frequent consumer heartbeating, which can overwhelm broker resources.

Type: int — **Default**: 6000 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

inter.broker.listener.name: Name of listener used for communication between brokers. If this is unset, the lis to set this and security.inter.broker.protocol properties at the same time.

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

inter.broker.protocol.version: Specify which version of the inter-broker protocol will be used. This is typically of some valid values are: 0.8.0, 0.8.1, 0.8.1.1, 0.8.2, 0.8.2.0, 0.8.2.1, 0.9.0.0, 0.9.0.1 Check ApiVersion for the f

Type: string — **Default**: 2.4-IV1
— **Valid Values**: [0.8.0, 0.8.1, 0.8.2, 0.9.0, 0.10.0-IV0, 0.10.0-IV1, 0.10.1-IV0, 0.10.1-IV1, 0.10.1-IV2, 0.10.2- IV1, 2.1-IV0, 2.1-IV1, 2.1-IV2, 2.2-IV0, 2.2-IV1, 2.3-IV0, 2.3-IV1, 2.4-IV0, 2.4-IV1]
— **Importance**: medium — **Update Mode**: read-only

---

log.cleaner.backoff.ms: The amount of time to sleep when there are no logs to clean

Type: long — **Default**: 15000 — **Valid Values**: [0,...] — **Importance**: medium — **Update Mode**: cluster-wic

---

log.cleaner.dedupe.buffer.size: The total memory used for log deduplication across all cleaner threads

Type: long — **Default**: 134217728 — **Valid Values**: — **Importance**: medium — **Update Mode**: cluster-wic

---

log.cleaner.delete.retention.ms: How long are delete records retained?

Type: long — **Default**: 86400000 — **Valid Values**: — **Importance**: medium — **Update Mode**: cluster-wide

---

log.cleaner.enable: Enable the log cleaner process to run on the server. Should be enabled if using any topics topic. If disabled those topics will not be compacted and continually grow in size.

Type: boolean — **Default**: true — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

log.cleaner.io.buffer.load.factor: Log cleaner dedupe buffer load factor. The percentage full the dedupe buffe once but will lead to more hash collisions

Type: double — **Default**: 0.9 — **Valid Values**: — **Importance**: medium — **Update Mode**: cluster-wide

---

log.cleaner.io.buffer.size: The total memory used for log cleaner I/O buffers across all cleaner threads

Type: int — **Default**: 524288 — **Valid Values**: [0,...] — **Importance**: medium — **Update Mode**: cluster-wid

---

log.cleaner.io.max.bytes.per.second: The log cleaner will be throttled so that the sum of its read and write i/

Type: double — **Default**: 1.7976931348623157E308 — **Valid Values**: — **Importance**: medium — **Update**

---

log.cleaner.max.compaction.lag.ms: The maximum time a message will remain ineligible for compaction in

Type: long — **Default**: 9223372036854775807 — **Valid Values**: — **Importance**: medium — **Update Mod**

---

log.cleaner.min.cleanable.ratio: The minimum ratio of dirty log to total log for a log to eligible for cleaning. If log.cleaner.min.compaction.lag.ms configurations are also specified, then the log compactor considers the l threshold has been met and the log has had dirty (uncompacted) records for at least the log.cleaner.min.com (uncompacted) records for at most the log.cleaner.max.compaction.lag.ms period.

Type: double — **Default**: 0.5 — **Valid Values**: — **Importance**: medium — **Update Mode**: cluster-wide

---

log.cleaner.min.compaction.lag.ms: The minimum time a message will remain uncompacted in the log. Only

Type: long — **Default**: 0 — **Valid Values**: — **Importance**: medium — **Update Mode**: cluster-wide

---

log.cleaner.threads: The number of background threads to use for log cleaning

Type: int — **Default**: 1 — **Valid Values**: [0,...] — **Importance**: medium — **Update Mode**: cluster-wide

---

log.cleanup.policy: The default cleanup policy for segments beyond the retention window. A comma separat

Type: list — **Default**: delete — **Valid Values**: [compact, delete] — **Importance**: medium — **Update Mode**:

---

log.index.interval.bytes: The interval with which we add an entry to the offset index

Type: int — **Default**: 4096 — **Valid Values**: [0,...] — **Importance**: medium — **Update Mode**: cluster-wide

---

log.index.size.max.bytes: The maximum size in bytes of the offset index

Type: int — **Default**: 10485760 — **Valid Values**: [4,...] — **Importance**: medium — **Update Mode**: cluster-v

---

log.message.format.version: Specify the message format version the broker will use to append messages tc are: 0.8.2, 0.9.0.0, 0.10.0, check ApiVersion for more details. By setting a particular message format version, smaller or equal than the specified version. Setting this value incorrectly will cause consumers with older ver they don't understand.

Type: string — **Default**: 2.4-IV1
— **Valid Values**: [0.8.0, 0.8.1, 0.8.2, 0.9.0, 0.10.0-IV0, 0.10.0-IV1, 0.10.1-IV0, 0.10.1-IV1, 0.10.1-IV2, 0.10.2- IV1, 2.1-IV0, 2.1-IV1, 2.1-IV2, 2.2-IV0, 2.2-IV1, 2.3-IV0, 2.3-IV1, 2.4-IV0, 2.4-IV1]
— **Importance**: medium — **Update Mode**: read-only

---

log.message.timestamp.difference.max.ms: The maximum difference allowed between the timestamp whe message. If log.message.timestamp.type=CreateTime, a message will be rejected if the difference in timest log.message.timestamp.type=LogAppendTime.The maximum timestamp difference allowed should be no gı rolling.

Type: long — **Default**: 9223372036854775807 — **Valid Values**: — **Importance**: medium — **Update Mod**

---

log.message.timestamp.type: Define whether the timestamp in the message is message create time or log a `LogAppendTime`

Type: string — **Default**: CreateTime — **Valid Values**: [CreateTime, LogAppendTime] — **Importance**: med

---

log.preallocate: Should pre allocate file when create new segment? If you are using Kafka on Windows, you p

Type: boolean — **Default**: false — **Valid Values**: — **Importance**: medium — **Update Mode**: cluster-wide

---

log.retention.check.interval.ms: The frequency in milliseconds that the log cleaner checks whether any log is

Type: long — **Default**: 300000 — **Valid Values**: [1,...] — **Importance**: medium — **Update Mode**: read-only

---

max.connections: The maximum number of connections we allow in the broker at any time. This limit is appl max.connections.per.ip. Listener-level limits may also be configured by prefixing the config name with the lis `listener.name.internal.max.connections` . Broker-wide limit should be configured based on brok application requirements. New connections are blocked if either the listener or broker limit is reached. Conne limit is reached. The least recently used connection on another listener will be closed in this case.

Type: int — **Default**: 2147483647 — **Valid Values**: [0,...] — **Importance**: medium — **Update Mode**: cluste

---

max.connections.per.ip: The maximum number of connections we allow from each ip address. This can be s max.connections.per.ip.overrides property. New connections from the ip address are dropped if the limit is re

Type: int — **Default**: 2147483647 — **Valid Values**: [0,...] — **Importance**: medium — **Update Mode**: cluste

---

max.connections.per.ip.overrides: A comma-separated list of per-ip or hostname overrides to the default ma "hostName:100,127.0.0.1:200"

Type: string — **Default**: "" — **Valid Values**: — **Importance**: medium — **Update Mode**: cluster-wide

---

max.incremental.fetch.session.cache.slots: The maximum number of incremental fetch sessions that we w

Type: int — **Default**: 1000 — **Valid Values**: [0,...] — **Importance**: medium — **Update Mode**: read-only

---

num.partitions: The default number of log partitions per topic

Type: int — **Default**: 1 — **Valid Values**: [1,...] — **Importance**: medium — **Update Mode**: read-only

---

password.encoder.old.secret: The old secret that was used for encoding dynamically configured passwords. dynamically encoded passwords are decoded using this old secret and re-encoded using password.encoder.

Type: password — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

password.encoder.secret: The secret used for encoding dynamically configured passwords for this broker.

Type: password — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

principal.builder.class: The fully qualified name of a class that implements the KafkaPrincipalBuilder interfac authorization. This config also supports the deprecated PrincipalBuilder interface which was previously used the default behavior depends on the security protocol in use. For SSL authentication, the principal will be deri `ssl.principal.mapping.rules` applied on the distinguished name from the client certificate if one is principal name will be ANONYMOUS. For SASL authentication, the principal will be derived using the rules de GSSAPI is in use, and the SASL authentication ID for other mechanisms. For PLAINTEXT, the principal will be

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

producer.purgatory.purge.interval.requests: The purge interval (in number of requests) of the producer reque

Type: int — **Default**: 1000 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

queued.max.request.bytes: The number of queued bytes allowed before no more requests are read

Type: long — **Default**: -1 — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

replica.fetch.backoff.ms: The amount of time to sleep when fetch partition error occurs.

Type: int — **Default**: 1000 — **Valid Values**: [0,...] — **Importance**: medium — **Update Mode**: read-only

---

replica.fetch.max.bytes: The number of bytes of messages to attempt to fetch for each partition. This is not empty partition of the fetch is larger than this value, the record batch will still be returned to ensure that progr the broker is defined via `message.max.bytes` (broker config) or `max.message.bytes` (topic config).

Type: int — **Default**: 1048576 — **Valid Values**: [0,...] — **Importance**: medium — **Update Mode**: read-only

---

replica.fetch.response.max.bytes: Maximum bytes expected for the entire fetch response. Records are fetch partition of the fetch is larger than this value, the record batch will still be returned to ensure that progress ca maximum record batch size accepted by the broker is defined via `message.max.bytes` (broker config) o

Type: int — **Default**: 10485760 — **Valid Values**: [0,...] — **Importance**: medium — **Update Mode**: read-onl

---

replica.selector.class: The fully qualified class name that implements ReplicaSelector. This is used by the bro implementation that returns the leader.

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

reserved.broker.max.id: Max number that can be used for a broker.id

Type: int — **Default**: 1000 — **Valid Values**: [0,...] — **Importance**: medium — **Update Mode**: read-only

---

sasl.client.callback.handler.class: The fully qualified name of a SASL client callback handler class that imple

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

sasl.enabled.mechanisms: The list of SASL mechanisms enabled in the Kafka server. The list may contain ar GSSAPI is enabled by default.

Type: list — **Default**: GSSAPI — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

sasl.jaas.config: JAAS login context parameters for SASL connections in the format used by JAAS configura format for the value is: '`loginModuleClass controlFlag (optionName=optionValue)*;`'. For br mechanism name in lower-case. For example, listener.name.sasl_ssl.scram-sha-256.sasl.jaas.config=com.e:

Type: password — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

sasl.kerberos.kinit.cmd: Kerberos kinit command path.

Type: string — **Default**: /usr/bin/kinit — **Valid Values**: — **Importance**: medium — **Update Mode**: per-bro

---

sasl.kerberos.min.time.before.relogin: Login thread sleep time between refresh attempts.

Type: long — **Default**: 60000 — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

sasl.kerberos.principal.to.local.rules: A list of rules for mapping from principal names to short names (typic and the first rule that matches a principal name is used to map it to a short name. Any later rules in the list ar {username}/{hostname}@{REALM} are mapped to {username}. For more details on the format please see se if an extension of KafkaPrincipalBuilder is provided by the `principal.builder.class` configuration.

Type: list — **Default**: DEFAULT — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

sasl.kerberos.service.name: The Kerberos principal name that Kafka runs as. This can be defined either in K

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

sasl.kerberos.ticket.renew.jitter: Percentage of random jitter added to the renewal time.

Type: double — **Default**: 0.05 — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

sasl.kerberos.ticket.renew.window.factor: Login thread will sleep until the specified window factor of time fr will try to renew the ticket.

Type: double — **Default**: 0.8 — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

sasl.login.callback.handler.class: The fully qualified name of a SASL login callback handler class that implen callback handler config must be prefixed with listener prefix and SASL mechanism name in lower-case. For e 256.sasl.login.callback.handler.class=com.example.CustomScramLoginCallbackHandler

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

sasl.login.class: The fully qualified name of a class that implements the Login interface. For brokers, login co name in lower-case. For example, listener.name.sasl_ssl.scram-sha-256.sasl.login.class=com.example.Custo

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

sasl.login.refresh.buffer.seconds: The amount of buffer time before credential expiration to maintain when r occur closer to expiration than the number of buffer seconds then the refresh will be moved up to maintain a and 3600 (1 hour); a default value of 300 (5 minutes) is used if no value is specified. This value and sasl.logi the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

Type: short — **Default**: 300 — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

sasl.login.refresh.min.period.seconds: The desired minimum time for the login refresh thread to wait before and 900 (15 minutes); a default value of 60 (1 minute) is used if no value is specified. This value and sasl.log remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

Type: short — **Default**: 60 — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

sasl.login.refresh.window.factor: Login refresh thread will sleep until the specified window factor relative to refresh the credential. Legal values are between 0.5 (50%) and 1.0 (100%) inclusive; a default value of 0.8 (80 OAUTHBEARER.

Type: double — **Default**: 0.8 — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

sasl.login.refresh.window.jitter: The maximum amount of random jitter relative to the credential's lifetime th between 0 and 0.25 (25%) inclusive; a default value of 0.05 (5%) is used if no value is specified. Currently app

Type: double — **Default**: 0.05 — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

sasl.mechanism.inter.broker.protocol: SASL mechanism used for inter-broker communication. Default is GS

Type: string — **Default**: GSSAPI — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

sasl.server.callback.handler.class: The fully qualified name of a SASL server callback handler class that imp callback handlers must be prefixed with listener prefix and SASL mechanism name in lower-case. For examp listener.name.sasl_ssl.plain.sasl.server.callback.handler.class=com.example.CustomPlainCallbackHandler.

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

security.inter.broker.protocol: Security protocol used to communicate between brokers. Valid values are: PL and inter.broker.listener.name properties at the same time.

Type: string — **Default**: PLAINTEXT — **Valid Values**: — **Importance**: medium — **Update Mode**: read-only

---

ssl.cipher.suites: A list of cipher suites. This is a named combination of authentication, encryption, MAC and a network connection using TLS or SSL network protocol. By default all the available cipher suites are suppo

Type: list — **Default**: "" — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

ssl.client.auth: Configures kafka broker to request client authentication. The following settings are common:

- `ssl.client.auth=required` If set to required client authentication is required.
- `ssl.client.auth=requested` This means client authentication is optional. unlike requested , if this information about itself
- `ssl.client.auth=none` This means client authentication is not needed.

Type: string — **Default**: none — **Valid Values**: [required, requested, none] — **Importance**: medium — **Up**

---

ssl.enabled.protocols: The list of protocols enabled for SSL connections.

Type: list — **Default**: TLSv1.2,TLSv1.1,TLSv1 — **Valid Values**: — **Importance**: medium — **Update Mode**:

---

ssl.key.password: The password of the private key in the key store file. This is optional for client.

Type: password — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

ssl.keymanager.algorithm: The algorithm used by key manager factory for SSL connections. Default value is Machine.

Type: string — **Default**: SunX509 — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

ssl.keystore.location: The location of the key store file. This is optional for client and can be used for two-wa

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

ssl.keystore.password: The store password for the key store file. This is optional for client and only needed i

Type: password — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

ssl.keystore.type: The file format of the key store file. This is optional for client.

Type: string — **Default**: JKS — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

ssl.protocol: The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for most TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to know

Type: string — **Default**: TLS — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

ssl.provider: The name of the security provider used for SSL connections. Default value is the default securit

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

ssl.trustmanager.algorithm: The algorithm used by trust manager factory for SSL connections. Default value Virtual Machine.

Type: string — **Default**: PKIX — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

ssl.truststore.location: The location of the trust store file.

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

ssl.truststore.password: The password for the trust store file. If a password is not set access to the truststor

Type: password — **Default**: null — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

ssl.truststore.type: The file format of the trust store file.

Type: string — **Default**: JKS — **Valid Values**: — **Importance**: medium — **Update Mode**: per-broker

---

alter.config.policy.class.name: The alter configs policy class that should be used for validation. The class sh `org.apache.kafka.server.policy.AlterConfigPolicy` interface.

Type: class — **Default**: null — **Valid Values**: — **Importance**: low — **Update Mode**: read-only

---

alter.log.dirs.replication.quota.window.num: The number of samples to retain in memory for alter log dirs re

Type: int — **Default**: 11 — **Valid Values**: [1,...] — **Importance**: low — **Update Mode**: read-only

---

alter.log.dirs.replication.quota.window.size.seconds: The time span of each sample for alter log dirs replica

Type: int — **Default**: 1 — **Valid Values**: [1,...] — **Importance**: low — **Update Mode**: read-only

---

authorizer.class.name: The fully qualified name of a class that implements sorg.apache.kafka.server.authori authorization. This config also supports authorizers that implement the deprecated kafka.security.auth.Auth

Type: string — **Default**: "" — **Valid Values**: — **Importance**: low — **Update Mode**: read-only

---

client.quota.callback.class: The fully qualified name of a class that implements the ClientQuotaCallback inte requests. By default, , or quotas stored in ZooKeeper are applied. For any given request, the most specific qu of the request is applied.

Type: class — **Default**: null — **Valid Values**: — **Importance**: low — **Update Mode**: read-only

---

connection.failed.authentication.delay.ms: Connection close delay on failed authentication: this is the time (authentication failure. This must be configured to be less than connections.max.idle.ms to prevent connectic

Type: int — **Default**: 100 — **Valid Values**: [0,...] — **Importance**: low — **Update Mode**: read-only

---

create.topic.policy.class.name: The create topic policy class that should be used for validation. The class sh `org.apache.kafka.server.policy.CreateTopicPolicy` interface.

Type: class — **Default**: null — **Valid Values**: — **Importance**: low — **Update Mode**: read-only

---

delegation.token.expiry.check.interval.ms: Scan interval to remove expired delegation tokens.

Type: long — **Default**: 3600000 — **Valid Values**: [1,...] — **Importance**: low — **Update Mode**: read-only

---

kafka.metrics.polling.interval.secs: The metrics polling interval (in seconds) which can be used in kafka.met

Type: int — **Default**: 10 — **Valid Values**: [1,...] — **Importance**: low — **Update Mode**: read-only

---

kafka.metrics.reporters: A list of classes to use as Yammer metrics custom reporters. The reporters should a client wants to expose JMX operations on a custom reporter, the custom reporter needs to additionally imp `kafka.metrics.KafkaMetricsReporterMBean` trait so that the registered MBean is compliant with t

Type: list — **Default**: "" — **Valid Values**: — **Importance**: low — **Update Mode**: read-only

---

listener.security.protocol.map: Map between listener names and security protocols. This must be defined for IP. For example, internal and external traffic can be separated even if SSL is required for both. Concretely, the and this property as: `INTERNAL:SSL,EXTERNAL:SSL`. As shown, key and value are separated by a colon and only appear once in the map. Different security (SSL and SASL) settings can be configured for each listener b the config name. For example, to set a different keystore for the INTERNAL listener, a config with name `lis` set. If the config for the listener name is not set, the config will fallback to the generic config (i.e. `ssl.keys`

Type: string — **Default**: PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:S — **Update Mode**: per-broker

---

log.message.downconversion.enable: This configuration controls whether down-conversion of message for `false` , broker will not perform down-conversion for consumers expecting an older message format. The b consume requests from such older clients. This configurationdoes not apply to any message format convers

Type: boolean — **Default**: true — **Valid Values**: — **Importance**: low — **Update Mode**: cluster-wide

---

metric.reporters: A list of classes to use as metrics reporters. Implementing the `org.apache.kafka.com` classes that will be notified of new metric creation. The JmxReporter is always included to register JMX stati

Type: list — **Default**: "" — **Valid Values**: — **Importance**: low — **Update Mode**: cluster-wide

---

metrics.num.samples: The number of samples maintained to compute metrics.

Type: int — **Default**: 2 — **Valid Values**: [1,...] — **Importance**: low — **Update Mode**: read-only

---

metrics.recording.level: The highest recording level for metrics.

Type: string — **Default**: INFO — **Valid Values**: — **Importance**: low — **Update Mode**: read-only

---

metrics.sample.window.ms: The window of time a metrics sample is computed over.

Type: long — **Default**: 30000 — **Valid Values**: [1,...] — **Importance**: low — **Update Mode**: read-only

---

password.encoder.cipher.algorithm: The Cipher algorithm used for encoding dynamically configured passwo

Type: string — **Default**: AES/CBC/PKCS5Padding — **Valid Values**: — **Importance**: low — **Update Mode**:

---

password.encoder.iterations: The iteration count used for encoding dynamically configured passwords.

Type: int — **Default**: 4096 — **Valid Values**: [1024,...] — **Importance**: low — **Update Mode**: read-only

---

password.encoder.key.length: The key length used for encoding dynamically configured passwords.

Type: int — **Default**: 128 — **Valid Values**: [8,...] — **Importance**: low — **Update Mode**: read-only

---

password.encoder.keyfactory.algorithm: The SecretKeyFactory algorithm used for encoding dynamically cor available and PBKDF2WithHmacSHA1 otherwise.

Type: string — **Default**: null — **Valid Values**: — **Importance**: low — **Update Mode**: read-only

---

quota.window.num: The number of samples to retain in memory for client quotas

Type: int — **Default**: 11 — **Valid Values**: [1,...] — **Importance**: low — **Update Mode**: read-only

---

quota.window.size.seconds: The time span of each sample for client quotas

Type: int — **Default**: 1 — **Valid Values**: [1,...] — **Importance**: low — **Update Mode**: read-only

---

replication.quota.window.num: The number of samples to retain in memory for replication quotas

Type: int — **Default**: 11 — **Valid Values**: [1,...] — **Importance**: low — **Update Mode**: read-only

---

replication.quota.window.size.seconds: The time span of each sample for replication quotas

Type: int — **Default**: 1 — **Valid Values**: [1,...] — **Importance**: low — **Update Mode**: read-only

---

security.providers: A list of configurable creator classes each returning a provider implementing security algo `org.apache.kafka.common.security.auth.SecurityProviderCreator` interface.

Type: string — **Default**: null — **Valid Values**: — **Importance**: low — **Update Mode**: read-only

---

ssl.endpoint.identification.algorithm: The endpoint identification algorithm to validate server hostname usin

Type: string — **Default**: https — **Valid Values**: — **Importance**: low — **Update Mode**: per-broker

---

ssl.principal.mapping.rules: A list of rules for mapping from distinguished name from the client certificate tc matches a principal name is used to map it to a short name. Any later rules in the list are ignored. By default, For more details on the format please see [security authorization and acls](). Note that this configuration is ignc `principal.builder.class` configuration.

Type: string — **Default**: DEFAULT — **Valid Values**: — **Importance**: low — **Update Mode**: read-only

---

ssl.secure.random.implementation: The SecureRandom PRNG implementation to use for SSL cryptography

Type: string — **Default**: null — **Valid Values**: — **Importance**: low — **Update Mode**: per-broker

---

transaction.abort.timed.out.transaction.cleanup.interval.ms: The interval at which to rollback transactions t

Type: int — **Default**: 60000 — **Valid Values**: [1,...] — **Importance**: low — **Update Mode**: read-only

---

transaction.remove.expired.transaction.cleanup.interval.ms: The interval at which to remove transactions th passing

Type: int — **Default**: 3600000 — **Valid Values**: [1,...] — **Importance**: low — **Update Mode**: read-only

---

zookeeper.sync.time.ms: How far a ZK follower can be behind a ZK leader

Type: int — **Default**: 2000 — **Valid Values**: — **Importance**: low — **Update Mode**: read-only

---

More details about broker configuration can be found in the scala class `kafka.server.KafkaConfig`.

3.1.1 Updating Broker Configs

From Kafka version 1.1 onwards, some of the broker configs can be updated without restarting the broker. See the update mode of each broker config.

- `read-only` : Requires a broker restart for update
- `per-broker` : May be updated dynamically for each broker
- `cluster-wide` : May be updated dynamically as a cluster-wide default. May also be updated as a per-broke

To alter the current broker configs for broker id 0 (for example, the number of log cleaner threads):

```
> bin/kafka-configs.sh --bootstrap-server localhost:9092 --entity-type brokers --enti
```

To describe the current dynamic broker configs for broker id 0:

```
> bin/kafka-configs.sh --bootstrap-server localhost:9092 --entity-type brokers --enti
```

To delete a config override and revert to the statically configured or default value for broker id 0 (for example, the r

```
> bin/kafka-configs.sh --bootstrap-server localhost:9092 --entity-type brokers --enti
```

Some configs may be configured as a cluster-wide default to maintain consistent values across the whole cluster. For example, to update log cleaner threads on all brokers:

```
> bin/kafka-configs.sh --bootstrap-server localhost:9092 --entity-type brokers --enti
```

To describe the currently configured dynamic cluster-wide default configs:

```
> bin/kafka-configs.sh --bootstrap-server localhost:9092 --entity-type brokers --enti
```

All configs that are configurable at cluster level may also be configured at per-broker level (e.g. for testing). If a co
precedence is used:

- Dynamic per-broker config stored in ZooKeeper
- Dynamic cluster-wide default config stored in ZooKeeper
- Static broker config from `server.properties`
- Kafka default, see broker configs

Updating Password Configs Dynamically

Password config values that are dynamically updated are encrypted before storing in ZooKeeper. The broker confi
`server.properties` to enable dynamic update of password configs. The secret may be different on different

The secret used for password encoding may be rotated with a rolling restart of brokers. The old secret used for en
static broker config `password.encoder.old.secret` and the new secret must be provided in `password.e`
ZooKeeper will be re-encoded with the new secret when the broker starts up.

In Kafka 1.1.x, all dynamically updated password configs must be provided in every alter request when updating co
not being altered. This constraint will be removed in a future release.

Updating Password Configs in ZooKeeper Before Starting Brokers

From Kafka 2.0.0 onwards, `kafka-configs.sh` enables dynamic broker configs to be updated using ZooKeep
password configs to be stored in encrypted form, avoiding the need for clear passwords in `server.propertie`
be specified if any password configs are included in the alter command. Additional encryption parameters may als
ZooKeeper. For example, to store SSL key password for listener `INTERNAL` on broker 0:

```
> bin/kafka-configs.sh --zookeeper localhost:2181 --entity-type brokers --entity-name
  'listener.name.internal.ssl.key.password=key-password,password.encoder.secret=secre
```

The configuration `listener.name.internal.ssl.key.password` will be persisted in ZooKeeper in encryp
and iterations are not persisted in ZooKeeper.

Updating SSL Keystore of an Existing Listener

Brokers may be configured with SSL keystores with short validity periods to reduce the risk of compromised certifi
the broker. The config name must be prefixed with the listener prefix `listener.name.{listenerName}.` so
following configs may be updated in a single alter request at per-broker level:

- `ssl.keystore.type`
- `ssl.keystore.location`
- `ssl.keystore.password`
- `ssl.key.password`

If the listener is the inter-broker listener, the update is allowed only if the new keystore is trusted by the truststore c
performed on the keystore by the broker. Certificates must be signed by the same certificate authority that signed

Updating SSL Truststore of an Existing Listener

Broker truststores may be updated dynamically without restarting the broker to add or remove certificates. Update
The config name must be prefixed with the listener prefix `listener.name.{listenerName}.` so that only th
configs may be updated in a single alter request at per-broker level:

- `ssl.truststore.type`
- `ssl.truststore.location`
- `ssl.truststore.password`

If the listener is the inter-broker listener, the update is allowed only if the existing keystore for that listener is truste
performed by the broker before the update. Removal of CA certificates used to sign client certificates from the nev

Updating Default Topic Configuration

Default topic configuration options used by brokers may be updated without broker restart. The configs are applie
topic config. One or more of these configs may be overridden at cluster-default level used by all brokers.

- `log.segment.bytes`
- `log.roll.ms`
- `log.roll.hours`
- `log.roll.jitter.ms`
- `log.roll.jitter.hours`
- `log.index.size.max.bytes`
- `log.flush.interval.messages`
- `log.flush.interval.ms`
- `log.retention.bytes`
- `log.retention.ms`
- `log.retention.minutes`
- `log.retention.hours`
- `log.index.interval.bytes`
- `log.cleaner.delete.retention.ms`
- `log.cleaner.min.compaction.lag.ms`
- `log.cleaner.max.compaction.lag.ms`
- `log.cleaner.min.cleanable.ratio`
- `log.cleanup.policy`
- `log.segment.delete.delay.ms`
- `unclean.leader.election.enable`
- `min.insync.replicas`
- `max.message.bytes`
- `compression.type`
- `log.preallocate`
- `log.message.timestamp.type`
- `log.message.timestamp.difference.max.ms`

From Kafka version 2.0.0 onwards, unclean leader election is automatically enabled by the controller when the cor updated. In Kafka version 1.1.x, changes to `unclean.leader.election.enable` take effect only when a ne running:

```
> bin/zookeeper-shell.sh localhost
rmr /controller
```

Updating Log Cleaner Configs

Log cleaner configs may be updated dynamically at cluster-default level used by all brokers. The changes take effe configs may be updated:

- `log.cleaner.threads`
- `log.cleaner.io.max.bytes.per.second`
- `log.cleaner.dedupe.buffer.size`
- `log.cleaner.io.buffer.size`
- `log.cleaner.io.buffer.load.factor`
- `log.cleaner.backoff.ms`

Updating Thread Configs

The size of various thread pools used by the broker may be updated dynamically at cluster-default level used by al `2` to `currentSize * 2` to ensure that config updates are handled gracefully.

- `num.network.threads`
- `num.io.threads`
- `num.replica.fetchers`
- `num.recovery.threads.per.data.dir`
- `log.cleaner.threads`
- `background.threads`

Updating ConnectionQuota Configs

The maximum number of connections allowed for a given IP/host by the broker may be updated dynamically at clu new connection creations and the existing connections count will be taken into account by the new limits.

- `max.connections.per.ip`
- `max.connections.per.ip.overrides`

Adding and Removing Listeners

Listeners may be added or removed dynamically. When a new listener is added, security configs of the listener mu `listener.name.{listenerName}.` . If the new listener uses SASL, the JAAS configuration of the listener mu `sasl.jaas.config` with the listener and mechanism prefix. See JAAS configuration for Kafka brokers for det

In Kafka version 1.1.x, the listener used by the inter-broker listener may not be updated dynamically. To update the added on all brokers without restarting the broker. A rolling restart is then required to update `inter.broker.li`

In addition to all the security configs of new listeners, the following configs may be updated dynamically at per-bro

- `listeners`
- `advertised.listeners`
- `listener.security.protocol.map`

Inter-broker listener must be configured using the static broker configuration `inter.broker.listener.name`

3.2 Topic-Level Configs

Configurations pertinent to topics have both a server default as well an optional per-topic override. If no per-topic be set at topic creation time by giving one or more `--config` options. This example creates a topic named *my-*

```
> bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic my-topic --p
    --replication-factor 1 --config max.message.bytes=64000 --config flush.messages=1
```

Overrides can also be changed or set later using the alter configs command. This example updates the max mess

```
> bin/kafka-configs.sh --zookeeper localhost:2181 --entity-type topics --entity-name
    --alter --add-config max.message.bytes=128000
```

To check overrides set on the topic you can do

```
> bin/kafka-configs.sh --zookeeper localhost:2181 --entity-type topics --entity-name
```

To remove an override you can do

```
> bin/kafka-configs.sh --zookeeper localhost:2181  --entity-type topics --entity-name
    --alter --delete-config max.message.bytes
```

The following are the topic-level configurations. The server's default configuration for this property is given under t value only applies to a topic if it does not have an explicit topic config override.

cleanup.policy: A string that is either "delete" or "compact" or both. This string designates the retention policy discard old segments when their retention time or size limit has been reached. The "compact" setting will ena

Type: list — **Default**: delete — **Valid Values**: [compact, delete] — **Server Default Property**: log.cleanup.po

---

compression.type: Specify the final compression type for a given topic. This configuration accepts the stand accepts 'uncompressed' which is equivalent to no compression; and 'producer' which means retain the origin

Type: string — **Default**: producer — **Valid Values**: [uncompressed, zstd, lz4, snappy, gzip, producer] — **S**

---

delete.retention.ms: The amount of time to retain delete tombstone markers for log compacted topics. This complete a read if they begin from offset 0 to ensure that they get a valid snapshot of the final stage (otherw scan).

Type: long — **Default**: 86400000 — **Valid Values**: [0,...] — **Server Default Property**: log.cleaner.delete.ret

---

file.delete.delay.ms: The time to wait before deleting a file from the filesystem

Type: long — **Default**: 60000 — **Valid Values**: [0,...] — **Server Default Property**: log.segment.delete.delay

---

flush.messages: This setting allows specifying an interval at which we will force an fsync of data written to t
message; if it were 5 we would fsync after every five messages. In general we recommend you not set this ar
background flush capabilities as it is more efficient. This setting can be overridden on a per-topic basis (see t

Type: long — **Default**: 9223372036854775807 — **Valid Values**: [0,...] — **Server Default Property**: log.flus

---

flush.ms: This setting allows specifying a time interval at which we will force an fsync of data written to the l
ms had passed. In general we recommend you not set this and use replication for durability and allow the op
efficient.

Type: long — **Default**: 9223372036854775807 — **Valid Values**: [0,...] — **Server Default Property**: log.flus

---

follower.replication.throttled.replicas: A list of replicas for which log replication should be throttled on the fo
[PartitionId]:[BrokerId],[PartitionId]:[BrokerId]:... or alternatively the wildcard '*' can be used to throttle all replic

Type: list — **Default**: "" — **Valid Values**: [partitionId]:[brokerId],[partitionId]:[brokerId],... — **Server Default**
— **Importance**: medium

---

index.interval.bytes: This setting controls how frequently Kafka adds an index entry to its offset index. The d
bytes. More indexing allows reads to jump closer to the exact position in the log but makes the index larger. Y

Type: int — **Default**: 4096 — **Valid Values**: [0,...] — **Server Default Property**: log.index.interval.bytes — I

---

leader.replication.throttled.replicas: A list of replicas for which log replication should be throttled on the leac
[PartitionId]:[BrokerId],[PartitionId]:[BrokerId]:... or alternatively the wildcard '*' can be used to throttle all replic

Type: list — **Default**: "" — **Valid Values**: [partitionId]:[brokerId],[partitionId]:[brokerId],... — **Server Default**
— **Importance**: medium

---

max.compaction.lag.ms: The maximum time a message will remain ineligible for compaction in the log. Only

Type: long — **Default**: 9223372036854775807 — **Valid Values**: [1,...] — **Server Default Property**: log.clea

---

max.message.bytes: The largest record batch size allowed by Kafka. If this is increased and there are consu
increased so that the they can fetch record batches this large. In the latest message format version, records
message format versions, uncompressed records are not grouped into batches and this limit only applies to

Type: int — **Default**: 1000012 — **Valid Values**: [0,...] — **Server Default Property**: message.max.bytes —

---

message.format.version: Specify the message format version the broker will use to append messages to the 0.8.2, 0.9.0.0, 0.10.0, check ApiVersion for more details. By setting a particular message format version, the u or equal than the specified version. Setting this value incorrectly will cause consumers with older versions to understand.

Type: string — **Default**: 2.4-IV1
— **Valid Values**: [0.8.0, 0.8.1, 0.8.2, 0.9.0, 0.10.0-IV0, 0.10.0-IV1, 0.10.1-IV0, 0.10.1-IV1, 0.10.1-IV2, 0.10.2- IV1, 2.1-IV0, 2.1-IV1, 2.1-IV2, 2.2-IV0, 2.2-IV1, 2.3-IV0, 2.3-IV1, 2.4-IV0, 2.4-IV1]
— **Server Default Property**: log.message.format.version — **Importance**: medium

---

message.timestamp.difference.max.ms: The maximum difference allowed between the timestamp when a b message. If message.timestamp.type=CreateTime, a message will be rejected if the difference in timestamp message.timestamp.type=LogAppendTime.

Type: long — **Default**: 9223372036854775807 — **Valid Values**: [0,...] — **Server Default Property**: log.me

---

message.timestamp.type: Define whether the timestamp in the message is message create time or log appe `LogAppendTime`

Type: string — **Default**: CreateTime — **Valid Values**: [CreateTime, LogAppendTime] — **Server Default Pr**

---

min.cleanable.dirty.ratio: This configuration controls how frequently the log compactor will attempt to clean avoid cleaning a log where more than 50% of the log has been compacted. This ratio bounds the maximum s log could be duplicates). A higher ratio will mean fewer, more efficient cleanings but will mean more wasted s min.compaction.lag.ms configurations are also specified, then the log compactor considers the log to be elig has been met and the log has had dirty (uncompacted) records for at least the min.compaction.lag.ms durat most the max.compaction.lag.ms period.

Type: double — **Default**: 0.5 — **Valid Values**: [0,...,1] — **Server Default Property**: log.cleaner.min.cleanab

---

min.compaction.lag.ms: The minimum time a message will remain uncompacted in the log. Only applicable

Type: long — **Default**: 0 — **Valid Values**: [0,...] — **Server Default Property**: log.cleaner.min.compaction.la

---

min.insync.replicas: When a producer sets acks to "all" (or "-1"), this configuration specifies the minimum nu considered successful. If this minimum cannot be met, then the producer will raise an exception (either NotE When used together, `min.insync.replicas` and `acks` allow you to enforce greater durability guarant factor of 3, set `min.insync.replicas` to 2, and produce with `acks` of "all". This will ensure that the p a write.

Type: int — **Default**: 1 — **Valid Values**: [1,...] — **Server Default Property**: min.insync.replicas — **Importa**

---

preallocate: True if we should preallocate the file on disk when creating a new log segment.

Type: boolean — **Default**: false — **Valid Values**: — **Server Default Property**: log.preallocate — **Importan**

---

retention.bytes: This configuration controls the maximum size a partition (which consists of log segments) c space if we are using the "delete" retention policy. By default there is no size limit only a time limit. Since this partitions to compute the topic retention in bytes.

Type: long — **Default**: -1 — **Valid Values**: — **Server Default Property**: log.retention.bytes — **Importance**:

---

retention.ms: This configuration controls the maximum time we will retain a log before we will discard old lo policy. This represents an SLA on how soon consumers must read their data. If set to -1, no time limit is appl

Type: long — **Default**: 604800000 — **Valid Values**: [-1,...] — **Server Default Property**: log.retention.ms —

---

segment.bytes: This configuration controls the segment file size for the log. Retention and cleaning is alway but less granular control over retention.

Type: int — **Default**: 1073741824 — **Valid Values**: [14,...] — **Server Default Property**: log.segment.bytes

---

segment.index.bytes: This configuration controls the size of the index that maps offsets to file positions. We generally should not need to change this setting.

Type: int — **Default**: 10485760 — **Valid Values**: [0,...] — **Server Default Property**: log.index.size.max.byte

---

segment.jitter.ms: The maximum random jitter subtracted from the scheduled segment roll time to avoid thu

Type: long — **Default**: 0 — **Valid Values**: [0,...] — **Server Default Property**: log.roll.jitter.ms — **Importanc**

---

segment.ms: This configuration controls the period of time after which Kafka will force the log to roll even if compact old data.

Type: long — **Default**: 604800000 — **Valid Values**: [1,...] — **Server Default Property**: log.roll.ms — **Impo**

---

unclean.leader.election.enable: Indicates whether to enable replicas not in the ISR set to be elected as leade

Type: boolean — **Default**: false — **Valid Values**: — **Server Default Property**: unclean.leader.election.enal

---

message.downconversion.enable: This configuration controls whether down-conversion of message format broker will not perform down-conversion for consumers expecting an older message format. The broker resp from such older clients. This configurationdoes not apply to any message format conversion that might be re

Type: boolean — **Default**: true — **Valid Values**: — **Server Default Property**: log.message.downconversio

---

3.3 Producer Configs

Below is the configuration of the producer:

key.serializer: Serializer class for key that implements the `org.apache.kafka.common.serializatio`

Type: class — **Default**: — **Valid Values**: — **Importance**: high

---

value.serializer: Serializer class for value that implements the `org.apache.kafka.common.serializa`

Type: class — **Default**: — **Valid Values**: — **Importance**: high

---

acks: The number of acknowledgments the producer requires the leader to have received before considering sent. The following settings are allowed:

- `acks=0` If set to zero then the producer will not wait for any acknowledgment from the server at all. The considered sent. No guarantee can be made that the server has received the record in this case, and the generally know of any failures). The offset given back for each record will always be set to `-1`.
- `acks=1` This will mean the leader will write the record to its local log but will respond without awaiting leader fail immediately after acknowledging the record but before the followers have replicated it then the
- `acks=all` This means the leader will wait for the full set of in-sync replicas to acknowledge the record one in-sync replica remains alive. This is the strongest available guarantee. This is equivalent to the acks=

Type: string — **Default**: 1 — **Valid Values**: [all, -1, 0, 1] — **Importance**: high

---

bootstrap.servers: A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. are specified here for bootstrapping—this list only impacts the initial hosts used to discover the full set of ser `host1:port1,host2:port2,...`. Since these servers are just used for the initial connection to discove this list need not contain the full set of servers (you may want more than one, though, in case a server is dow

Type: list — **Default**: "" — **Valid Values**: non-null string — **Importance**: high

---

buffer.memory: The total bytes of memory the producer can use to buffer records waiting to be sent to the se server the producer will block for `max.block.ms` after which it will throw an exception.

This setting should correspond roughly to the total memory the producer will use, but is not a hard bound sin additional memory will be used for compression (if compression is enabled) as well as for maintaining in-flig

Type: long — **Default**: 33554432 — **Valid Values**: [0,...] — **Importance**: high

---

compression.type: The compression type for all data generated by the producer. The default is none (i.e. no or `zstd`. Compression is of full batches of data, so the efficacy of batching will also impact the compressi

Type: string — **Default**: none — **Valid Values**: — **Importance**: high

---

retries: Setting a value greater than zero will cause the client to resend any record whose send fails with a po the client resent the record upon receiving the error. Allowing retries without setting `max.in.flight.req` ordering of records because if two batches are sent to a single partition, and the first fails and is retried but tl appear first. Note additionally that produce requests will be failed before the number of retries has been exha expires first before successful acknowledgement. Users should generally prefer to leave this config unset an

Type: int — **Default**: 2147483647 — **Valid Values**: [0,...,2147483647] — **Importance**: high

---

ssl.key.password: The password of the private key in the key store file. This is optional for client.

Type: password — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.keystore.location: The location of the key store file. This is optional for client and can be used for two-wa

Type: string — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.keystore.password: The store password for the key store file. This is optional for client and only needed i

Type: password — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.truststore.location: The location of the trust store file.

Type: string — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.truststore.password: The password for the trust store file. If a password is not set access to the truststor

Type: password — **Default**: null — **Valid Values**: — **Importance**: high

---

batch.size: The producer will attempt to batch records together into fewer requests whenever multiple record both the client and the server. This configuration controls the default batch size in bytes.

No attempt will be made to batch records larger than this size.

Requests sent to brokers will contain multiple batches, one for each partition with data available to be sent.

A small batch size will make batching less common and may reduce throughput (a batch size of zero will dis bit more wastefully as we will always allocate a buffer of the specified batch size in anticipation of additional

Type: int — **Default**: 16384 — **Valid Values**: [0,...] — **Importance**: medium

---

client.dns.lookup: Controls how the client uses DNS lookups. If set to `use_all_dns_ips` then, when the be attempted to connect to before failing the connection. Applies to both bootstrap and advertised servers. I `resolve_canonical_bootstrap_servers_only` each entry will be resolved and expanded into a list

Type: string — **Default**: default — **Valid Values**: [default, use_all_dns_ips, resolve_canonical_bootstrap_s

---

client.id: An id string to pass to the server when making requests. The purpose of this is to be able to track th application name to be included in server-side request logging.

Type: string — **Default**: "" — **Valid Values**: — **Importance**: medium

---

connections.max.idle.ms: Close idle connections after the number of milliseconds specified by this config.

Type: long — **Default**: 540000 — **Valid Values**: — **Importance**: medium

---

delivery.timeout.ms: An upper bound on the time to report success or failure after a call to `send()` returns sending, the time to await acknowledgement from the broker (if expected), and the time allowed for retriable earlier than this config if either an unrecoverable error is encountered, the retries have been exhausted, or the expiration deadline. The value of this config should be greater than or equal to the sum of `request.timec`

Type: int — **Default**: 120000 — **Valid Values**: [0,...] — **Importance**: medium

---

linger.ms: The producer groups together any records that arrive in between request transmissions into a sing records arrive faster than they can be sent out. However in some circumstances the client may want to reduc accomplishes this by adding a small amount of artificial delay—that is, rather than immediately sending out a other records to be sent so that the sends can be batched together. This can be thought of as analogous to N delay for batching: once we get `batch.size` worth of records for a partition it will be sent immediately re bytes accumulated for this partition we will 'linger' for the specified time waiting for more records to show up for example, would have the effect of reducing the number of requests sent but would add up to 5ms of laten

Type: long — **Default**: 0 — **Valid Values**: [0,...] — **Importance**: medium

---

max.block.ms: The configuration controls how long `KafkaProducer.send()` and `KafkaProducer.p` either because the buffer is full or metadata unavailable.Blocking in the user-supplied serializers or partitione

Type: long — **Default**: 60000 — **Valid Values**: [0,...] — **Importance**: medium

---

max.request.size: The maximum size of a request in bytes. This setting will limit the number of record batch requests. This is also effectively a cap on the maximum record batch size. Note that the server has its own c

Type: int — **Default**: 1048576 — **Valid Values**: [0,...] — **Importance**: medium

---

partitioner.class: Partitioner class that implements the `org.apache.kafka.clients.producer.Part`

Type: class — **Default**: org.apache.kafka.clients.producer.internals.DefaultPartitioner — **Valid Values**: —

---

receive.buffer.bytes: The size of the TCP receive buffer (SO_RCVBUF) to use when reading data. If the value

Type: int — **Default**: 32768 — **Valid Values**: [-1,...] — **Importance**: medium

---

request.timeout.ms: The configuration controls the maximum amount of time the client will wait for the resp timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted. This configuration) to reduce the possibility of message duplication due to unnecessary producer retries.

Type: int — **Default**: 30000 — **Valid Values**: [0,...] — **Importance**: medium

---

sasl.client.callback.handler.class: The fully qualified name of a SASL client callback handler class that imple

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.jaas.config: JAAS login context parameters for SASL connections in the format used by JAAS configura format for the value is: '`loginModuleClass controlFlag (optionName=optionValue)*;`'. For br mechanism name in lower-case. For example, listener.name.sasl_ssl.scram-sha-256.sasl.jaas.config=com.e

Type: password — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.kerberos.service.name: The Kerberos principal name that Kafka runs as. This can be defined either in K

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.login.callback.handler.class: The fully qualified name of a SASL login callback handler class that implen callback handler config must be prefixed with listener prefix and SASL mechanism name in lower-case. For e 256.sasl.login.callback.handler.class=com.example.CustomScramLoginCallbackHandler

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.login.class: The fully qualified name of a class that implements the Login interface. For brokers, login co name in lower-case. For example, listener.name.sasl_ssl.scram-sha-256.sasl.login.class=com.example.Custo

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.mechanism: SASL mechanism used for client connections. This may be any mechanism for which a sec

Type: string — **Default**: GSSAPI — **Valid Values**: — **Importance**: medium

---

security.protocol: Protocol used to communicate with brokers. Valid values are: PLAINTEXT, SSL, SASL_PLAI

Type: string — **Default**: PLAINTEXT — **Valid Values**: — **Importance**: medium

---

send.buffer.bytes: The size of the TCP send buffer (SO_SNDBUF) to use when sending data. If the value is -1

Type: int — **Default**: 131072 — **Valid Values**: [-1,...] — **Importance**: medium

---

ssl.enabled.protocols: The list of protocols enabled for SSL connections.

Type: list — **Default**: TLSv1.2,TLSv1.1,TLSv1 — **Valid Values**: — **Importance**: medium

---

ssl.keystore.type: The file format of the key store file. This is optional for client.

Type: string — **Default**: JKS — **Valid Values**: — **Importance**: medium

---

ssl.protocol: The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for mos
TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to know

Type: string — **Default**: TLS — **Valid Values**: — **Importance**: medium

---

ssl.provider: The name of the security provider used for SSL connections. Default value is the default securit

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

ssl.truststore.type: The file format of the trust store file.

Type: string — **Default**: JKS — **Valid Values**: — **Importance**: medium

---

enable.idempotence: When set to 'true', the producer will ensure that exactly one copy of each message is w
etc., may write duplicates of the retried message in the stream. Note that enabling idempotence requires `ma`
or equal to 5, `retries` to be greater than 0 and `acks` must be 'all'. If these values are not explicitly set b
are set, a `ConfigException` will be thrown.

Type: boolean — **Default**: false — **Valid Values**: — **Importance**: low

---

interceptor.classes: A list of classes to use as interceptors. Implementing the `org.apache.kafka.clie`
to intercept (and possibly mutate) the records received by the producer before they are published to the Kafk

Type: list — **Default**: "" — **Valid Values**: non-null string — **Importance**: low

---

max.in.flight.requests.per.connection: The maximum number of unacknowledged requests the client will se
is set to be greater than 1 and there are failed sends, there is a risk of message re-ordering due to retries (i.e.

Type: int — **Default**: 5 — **Valid Values**: [1,...] — **Importance**: low

---

metadata.max.age.ms: The period of time in milliseconds after which we force a refresh of metadata even if
discover any new brokers or partitions.

Type: long — **Default**: 300000 — **Valid Values**: [0,...] — **Importance**: low

---

metric.reporters: A list of classes to use as metrics reporters. Implementing the `org.apache.kafka.com`
classes that will be notified of new metric creation. The JmxReporter is always included to register JMX stati

Type: list — **Default**: "" — **Valid Values**: non-null string — **Importance**: low

metrics.num.samples: The number of samples maintained to compute metrics.

Type: int — **Default**: 2 — **Valid Values**: [1,...] — **Importance**: low

metrics.recording.level: The highest recording level for metrics.

Type: string — **Default**: INFO — **Valid Values**: [INFO, DEBUG] — **Importance**: low

metrics.sample.window.ms: The window of time a metrics sample is computed over.

Type: long — **Default**: 30000 — **Valid Values**: [0,...] — **Importance**: low

reconnect.backoff.max.ms: The maximum amount of time in milliseconds to wait when reconnecting to a br
per host will increase exponentially for each consecutive connection failure, up to this maximum. After calcu
connection storms.

Type: long — **Default**: 1000 — **Valid Values**: [0,...] — **Importance**: low

reconnect.backoff.ms: The base amount of time to wait before attempting to reconnect to a given host. This
backoff applies to all connection attempts by the client to a broker.

Type: long — **Default**: 50 — **Valid Values**: [0,...] — **Importance**: low

retry.backoff.ms: The amount of time to wait before attempting to retry a failed request to a given topic parti
some failure scenarios.

Type: long — **Default**: 100 — **Valid Values**: [0,...] — **Importance**: low

sasl.kerberos.kinit.cmd: Kerberos kinit command path.

Type: string — **Default**: /usr/bin/kinit — **Valid Values**: — **Importance**: low

sasl.kerberos.min.time.before.relogin: Login thread sleep time between refresh attempts.

Type: long — **Default**: 60000 — **Valid Values**: — **Importance**: low

sasl.kerberos.ticket.renew.jitter: Percentage of random jitter added to the renewal time.

Type: double — **Default**: 0.05 — **Valid Values**: — **Importance**: low

sasl.kerberos.ticket.renew.window.factor: Login thread will sleep until the specified window factor of time fr
will try to renew the ticket.

Type: double — **Default**: 0.8 — **Valid Values**: — **Importance**: low

sasl.login.refresh.buffer.seconds: The amount of buffer time before credential expiration to maintain when r occur closer to expiration than the number of buffer seconds then the refresh will be moved up to maintain a and 3600 (1 hour); a default value of 300 (5 minutes) is used if no value is specified. This value and sasl.logir the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

Type: short — **Default**: 300 — **Valid Values**: [0,...,3600] — **Importance**: low

---

sasl.login.refresh.min.period.seconds: The desired minimum time for the login refresh thread to wait before and 900 (15 minutes); a default value of 60 (1 minute) is used if no value is specified. This value and sasl.log remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

Type: short — **Default**: 60 — **Valid Values**: [0,...,900] — **Importance**: low

---

sasl.login.refresh.window.factor: Login refresh thread will sleep until the specified window factor relative to t refresh the credential. Legal values are between 0.5 (50%) and 1.0 (100%) inclusive; a default value of 0.8 (80 OAUTHBEARER.

Type: double — **Default**: 0.8 — **Valid Values**: [0.5,...,1.0] — **Importance**: low

---

sasl.login.refresh.window.jitter: The maximum amount of random jitter relative to the credential's lifetime th between 0 and 0.25 (25%) inclusive; a default value of 0.05 (5%) is used if no value is specified. Currently app

Type: double — **Default**: 0.05 — **Valid Values**: [0.0,...,0.25] — **Importance**: low

---

security.providers: A list of configurable creator classes each returning a provider implementing security algo `org.apache.kafka.common.security.auth.SecurityProviderCreator` interface.

Type: string — **Default**: null — **Valid Values**: — **Importance**: low

---

ssl.cipher.suites: A list of cipher suites. This is a named combination of authentication, encryption, MAC and a network connection using TLS or SSL network protocol. By default all the available cipher suites are suppor

Type: list — **Default**: null — **Valid Values**: — **Importance**: low

---

ssl.endpoint.identification.algorithm: The endpoint identification algorithm to validate server hostname usin

Type: string — **Default**: https — **Valid Values**: — **Importance**: low

---

ssl.keymanager.algorithm: The algorithm used by key manager factory for SSL connections. Default value is Machine.

Type: string — **Default**: SunX509 — **Valid Values**: — **Importance**: low

---

ssl.secure.random.implementation: The SecureRandom PRNG implementation to use for SSL cryptography

Type: string — **Default**: null — **Valid Values**: — **Importance**: low

---

ssl.trustmanager.algorithm: The algorithm used by trust manager factory for SSL connections. Default value Virtual Machine.

Type: string — **Default**: PKIX — **Valid Values**: — **Importance**: low

---

transaction.timeout.ms: The maximum amount of time in ms that the transaction coordinator will wait for a aborting the ongoing transaction.If this value is larger than the transaction.max.timeout.ms setting in the brc error.

Type: int — **Default**: 60000 — **Valid Values**: — **Importance**: low

---

transactional.id: The TransactionalId to use for transactional delivery. This enables reliability semantics whic guarantee that transactions using the same TransactionalId have been completed prior to starting any new tı limited to idempotent delivery. Note that `enable.idempotence` must be enabled if a TransactionalId is c cannot be used. Note that, by default, transactions require a cluster of at least three brokers which is the reco this, by adjusting broker setting `transaction.state.log.replication.factor` .

Type: string — **Default**: null — **Valid Values**: non-empty string — **Importance**: low

---

3.4 Consumer Configs

Below is the configuration for the consumer:

key.deserializer: Deserializer class for key that implements the `org.apache.kafka.common.serializa`

Type: class — **Default**: — **Valid Values**: — **Importance**: high

---

value.deserializer: Deserializer class for value that implements the `org.apache.kafka.common.seria`

Type: class — **Default**: — **Valid Values**: — **Importance**: high

---

bootstrap.servers: A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. are specified here for bootstrapping—this list only impacts the initial hosts used to discover the full set of ser `host1:port1,host2:port2,...` . Since these servers are just used for the initial connection to discove this list need not contain the full set of servers (you may want more than one, though, in case a server is dow

Type: list — **Default**: "" — **Valid Values**: non-null string — **Importance**: high

---

fetch.min.bytes: The minimum amount of data the server should return for a fetch request. If insufficient dat accumulate before answering the request. The default setting of 1 byte means that fetch requests are answe request times out waiting for data to arrive. Setting this to something greater than 1 will cause the server to v server throughput a bit at the cost of some additional latency.

Type: int — **Default**: 1 — **Valid Values**: [0,...] — **Importance**: high

---

group.id: A unique string that identifies the consumer group this consumer belongs to. This property is requi functionality by using `subscribe(topic)` or the Kafka-based offset management strategy.

Type: string — **Default**: null — **Valid Values**: — **Importance**: high

---

heartbeat.interval.ms: The expected time between heartbeats to the consumer coordinator when using Kafk that the consumer's session stays active and to facilitate rebalancing when new consumers join or leave the `session.timeout.ms`, but typically should be set no higher than 1/3 of that value. It can be adjusted eve

Type: int — **Default**: 3000 — **Valid Values**: — **Importance**: high

---

max.partition.fetch.bytes: The maximum amount of data per-partition the server will return. Records are fetc non-empty partition of the fetch is larger than this limit, the batch will still be returned to ensure that the cons accepted by the broker is defined via `message.max.bytes` (broker config) or `max.message.bytes` (request size.

Type: int — **Default**: 1048576 — **Valid Values**: [0,...] — **Importance**: high

---

session.timeout.ms: The timeout used to detect client failures when using Kafka's group management facilit the broker. If no heartbeats are received by the broker before the expiration of this session timeout, then the l rebalance. Note that the value must be in the allowable range as configured in the broker configuration by `g` `group.max.session.timeout.ms`.

Type: int — **Default**: 10000 — **Valid Values**: — **Importance**: high

---

ssl.key.password: The password of the private key in the key store file. This is optional for client.

Type: password — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.keystore.location: The location of the key store file. This is optional for client and can be used for two-wa

Type: string — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.keystore.password: The store password for the key store file. This is optional for client and only needed i

Type: password — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.truststore.location: The location of the trust store file.

Type: string — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.truststore.password: The password for the trust store file. If a password is not set access to the truststo

Type: password — **Default**: null — **Valid Values**: — **Importance**: high

---

allow.auto.create.topics: Allow automatic topic creation on the broker when subscribing to or assigning a to the broker allows for it using `auto.create.topics.enable` broker configuration. This configuration must be set

Type: boolean — **Default**: true — **Valid Values**: — **Importance**: medium

---

auto.offset.reset: What to do when there is no initial offset in Kafka or if the current offset does not exist any

- 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.

Type: string — **Default**: latest — **Valid Values**: [latest, earliest, none] — **Importance**: medium

---

client.dns.lookup: Controls how the client uses DNS lookups. If set to `use_all_dns_ips` then, when the be attempted to connect to before failing the connection. Applies to both bootstrap and advertised servers. I `resolve_canonical_bootstrap_servers_only` each entry will be resolved and expanded into a list

Type: string — **Default**: default — **Valid Values**: [default, use_all_dns_ips, resolve_canonical_bootstrap_s

---

connections.max.idle.ms: Close idle connections after the number of milliseconds specified by this config.

Type: long — **Default**: 540000 — **Valid Values**: — **Importance**: medium

---

default.api.timeout.ms: Specifies the timeout (in milliseconds) for consumer APIs that could block. This con operations that do not explicitly accept a `timeout` parameter.

Type: int — **Default**: 60000 — **Valid Values**: [0,...] — **Importance**: medium

---

enable.auto.commit: If true the consumer's offset will be periodically committed in the background.

Type: boolean — **Default**: true — **Valid Values**: — **Importance**: medium

---

exclude.internal.topics: Whether internal topics matching a subscribed pattern should be excluded from the internal topic.

Type: boolean — **Default**: true — **Valid Values**: — **Importance**: medium

---

fetch.max.bytes: The maximum amount of data the server should return for a fetch request. Records are fetc first non-empty partition of the fetch is larger than this value, the record batch will still be returned to ensure t

absolute maximum. The maximum record batch size accepted by the broker is defined via `message.max.l`
Note that the consumer performs multiple fetches in parallel.

Type: int — **Default**: 52428800 — **Valid Values**: [0,...] — **Importance**: medium

---

group.instance.id: A unique identifier of the consumer instance provided by the end user. Only non-empty str
member, which means that only one instance with this ID is allowed in the consumer group at any time. This
group rebalances caused by transient unavailability (e.g. process restarts). If not set, the consumer will join t

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

isolation.level: Controls how to read messages written transactionally. If set to `read_committed`, consur
committed. If set to `read_uncommitted`' (the default), consumer.poll() will return all messages, even trar
messages will be returned unconditionally in either mode.

Messages will always be returned in offset order. Hence, in `read_committed` mode, consumer.poll() will
the one less than the offset of the first open transaction. In particular any messages appearing after messag
relevant transaction has been completed. As a result, `read_committed` consumers will not be able to rea

Further, when in `read_committed` the seekToEnd method will return the LSO

Type: string — **Default**: read_uncommitted — **Valid Values**: [read_committed, read_uncommitted] — **Im**

---

max.poll.interval.ms: The maximum delay between invocations of poll() when using consumer group manag
consumer can be idle before fetching more records. If poll() is not called before expiration of this timeout, th
in order to reassign the partitions to another member. For consumers using a non-null `group.instance.i`
reassigned. Instead, the consumer will stop sending heartbeats and partitions will be reassigned after expira
static consumer which has shutdown.

Type: int — **Default**: 300000 — **Valid Values**: [1,...] — **Importance**: medium

---

max.poll.records: The maximum number of records returned in a single call to poll().

Type: int — **Default**: 500 — **Valid Values**: [1,...] — **Importance**: medium

---

partition.assignment.strategy: A list of class names or class types, ordered by preference, of supported assi
client will use to distribute partition ownership amongst consumer instances when group management is us
`org.apache.kafka.clients.consumer.ConsumerPartitionAssignor` interface allows you to pl

Type: list — **Default**: class org.apache.kafka.clients.consumer.RangeAssignor — **Valid Values**: non-null s

---

receive.buffer.bytes: The size of the TCP receive buffer (SO_RCVBUF) to use when reading data. If the value

Type: int — **Default**: 65536 — **Valid Values**: [-1,...] — **Importance**: medium

---

request.timeout.ms: The configuration controls the maximum amount of time the client will wait for the resp timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.

Type: int — **Default**: 30000 — **Valid Values**: [0,...] — **Importance**: medium

---

sasl.client.callback.handler.class: The fully qualified name of a SASL client callback handler class that imple

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.jaas.config: JAAS login context parameters for SASL connections in the format used by JAAS configura format for the value is: '`loginModuleClass controlFlag (optionName=optionValue)*;`'. For br mechanism name in lower-case. For example, listener.name.sasl_ssl.scram-sha-256.sasl.jaas.config=com.e:

Type: password — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.kerberos.service.name: The Kerberos principal name that Kafka runs as. This can be defined either in K

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.login.callback.handler.class: The fully qualified name of a SASL login callback handler class that implem callback handler config must be prefixed with listener prefix and SASL mechanism name in lower-case. For e 256.sasl.login.callback.handler.class=com.example.CustomScramLoginCallbackHandler

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.login.class: The fully qualified name of a class that implements the Login interface. For brokers, login cc name in lower-case. For example, listener.name.sasl_ssl.scram-sha-256.sasl.login.class=com.example.Custo

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.mechanism: SASL mechanism used for client connections. This may be any mechanism for which a sec

Type: string — **Default**: GSSAPI — **Valid Values**: — **Importance**: medium

---

security.protocol: Protocol used to communicate with brokers. Valid values are: PLAINTEXT, SSL, SASL_PLAI

Type: string — **Default**: PLAINTEXT — **Valid Values**: — **Importance**: medium

---

send.buffer.bytes: The size of the TCP send buffer (SO_SNDBUF) to use when sending data. If the value is -1,

Type: int — **Default**: 131072 — **Valid Values**: [-1,...] — **Importance**: medium

---

ssl.enabled.protocols: The list of protocols enabled for SSL connections.

Type: list — **Default**: TLSv1.2,TLSv1.1,TLSv1 — **Valid Values**: — **Importance**: medium

---

ssl.keystore.type: The file format of the key store file. This is optional for client.

Type: string — **Default**: JKS — **Valid Values**: — **Importance**: medium

---

ssl.protocol: The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for most
TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to know

Type: string — **Default**: TLS — **Valid Values**: — **Importance**: medium

---

ssl.provider: The name of the security provider used for SSL connections. Default value is the default securit

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

ssl.truststore.type: The file format of the trust store file.

Type: string — **Default**: JKS — **Valid Values**: — **Importance**: medium

---

auto.commit.interval.ms: The frequency in milliseconds that the consumer offsets are auto-committed to Ka

Type: int — **Default**: 5000 — **Valid Values**: [0,...] — **Importance**: low

---

check.crcs: Automatically check the CRC32 of the records consumed. This ensures no on-the-wire or on-disk
overhead, so it may be disabled in cases seeking extreme performance.

Type: boolean — **Default**: true — **Valid Values**: — **Importance**: low

---

client.id: An id string to pass to the server when making requests. The purpose of this is to be able to track th
application name to be included in server-side request logging.

Type: string — **Default**: "" — **Valid Values**: — **Importance**: low

---

client.rack: A rack identifier for this client. This can be any string value which indicates where this client is ph

Type: string — **Default**: "" — **Valid Values**: — **Importance**: low

---

fetch.max.wait.ms: The maximum amount of time the server will block before answering the fetch request if
given by fetch.min.bytes.

Type: int — **Default**: 500 — **Valid Values**: [0,...] — **Importance**: low

---

interceptor.classes: A list of classes to use as interceptors. Implementing the `org.apache.kafka.clier`
to intercept (and possibly mutate) records received by the consumer. By default, there are no interceptors.

Type: list — **Default**: "" — **Valid Values**: non-null string — **Importance**: low

---

metadata.max.age.ms: The period of time in milliseconds after which we force a refresh of metadata even if discover any new brokers or partitions.

Type: long — **Default**: 300000 — **Valid Values**: [0,...] — **Importance**: low

---

metric.reporters: A list of classes to use as metrics reporters. Implementing the `org.apache.kafka.com` classes that will be notified of new metric creation. The JmxReporter is always included to register JMX stati

Type: list — **Default**: "" — **Valid Values**: non-null string — **Importance**: low

---

metrics.num.samples: The number of samples maintained to compute metrics.

Type: int — **Default**: 2 — **Valid Values**: [1,...] — **Importance**: low

---

metrics.recording.level: The highest recording level for metrics.

Type: string — **Default**: INFO — **Valid Values**: [INFO, DEBUG] — **Importance**: low

---

metrics.sample.window.ms: The window of time a metrics sample is computed over.

Type: long — **Default**: 30000 — **Valid Values**: [0,...] — **Importance**: low

---

reconnect.backoff.max.ms: The maximum amount of time in milliseconds to wait when reconnecting to a br per host will increase exponentially for each consecutive connection failure, up to this maximum. After calcu connection storms.

Type: long — **Default**: 1000 — **Valid Values**: [0,...] — **Importance**: low

---

reconnect.backoff.ms: The base amount of time to wait before attempting to reconnect to a given host. This backoff applies to all connection attempts by the client to a broker.

Type: long — **Default**: 50 — **Valid Values**: [0,...] — **Importance**: low

---

retry.backoff.ms: The amount of time to wait before attempting to retry a failed request to a given topic parti some failure scenarios.

Type: long — **Default**: 100 — **Valid Values**: [0,...] — **Importance**: low

---

sasl.kerberos.kinit.cmd: Kerberos kinit command path.

Type: string — **Default**: /usr/bin/kinit — **Valid Values**: — **Importance**: low

---

sasl.kerberos.min.time.before.relogin: Login thread sleep time between refresh attempts.

Type: long — **Default**: 60000 — **Valid Values**: — **Importance**: low

---

sasl.kerberos.ticket.renew.jitter: Percentage of random jitter added to the renewal time.

Type: double — **Default**: 0.05 — **Valid Values**: — **Importance**: low

---

sasl.kerberos.ticket.renew.window.factor: Login thread will sleep until the specified window factor of time fr will try to renew the ticket.

Type: double — **Default**: 0.8 — **Valid Values**: — **Importance**: low

---

sasl.login.refresh.buffer.seconds: The amount of buffer time before credential expiration to maintain when r occur closer to expiration than the number of buffer seconds then the refresh will be moved up to maintain a and 3600 (1 hour); a default value of 300 (5 minutes) is used if no value is specified. This value and sasl.logi the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

Type: short — **Default**: 300 — **Valid Values**: [0,...,3600] — **Importance**: low

---

sasl.login.refresh.min.period.seconds: The desired minimum time for the login refresh thread to wait before and 900 (15 minutes); a default value of 60 (1 minute) is used if no value is specified. This value and sasl.log remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

Type: short — **Default**: 60 — **Valid Values**: [0,...,900] — **Importance**: low

---

sasl.login.refresh.window.factor: Login refresh thread will sleep until the specified window factor relative to refresh the credential. Legal values are between 0.5 (50%) and 1.0 (100%) inclusive; a default value of 0.8 (8 OAUTHBEARER.

Type: double — **Default**: 0.8 — **Valid Values**: [0.5,...,1.0] — **Importance**: low

---

sasl.login.refresh.window.jitter: The maximum amount of random jitter relative to the credential's lifetime th between 0 and 0.25 (25%) inclusive; a default value of 0.05 (5%) is used if no value is specified. Currently app

Type: double — **Default**: 0.05 — **Valid Values**: [0.0,...,0.25] — **Importance**: low

---

security.providers: A list of configurable creator classes each returning a provider implementing security alg `org.apache.kafka.common.security.auth.SecurityProviderCreator` interface.

Type: string — **Default**: null — **Valid Values**: — **Importance**: low

---

ssl.cipher.suites: A list of cipher suites. This is a named combination of authentication, encryption, MAC and a network connection using TLS or SSL network protocol. By default all the available cipher suites are suppo

Type: list — **Default**: null — **Valid Values**: — **Importance**: low

---

ssl.endpoint.identification.algorithm: The endpoint identification algorithm to validate server hostname usin

Type: string — **Default**: https — **Valid Values**: — **Importance**: low

---

ssl.keymanager.algorithm: The algorithm used by key manager factory for SSL connections. Default value is Machine.

Type: string — **Default**: SunX509 — **Valid Values**: — **Importance**: low

---

ssl.secure.random.implementation: The SecureRandom PRNG implementation to use for SSL cryptography

Type: string — **Default**: null — **Valid Values**: — **Importance**: low

---

ssl.trustmanager.algorithm: The algorithm used by trust manager factory for SSL connections. Default value Virtual Machine.

Type: string — **Default**: PKIX — **Valid Values**: — **Importance**: low

---

3.5 Kafka Connect Configs

Below is the configuration of the Kafka Connect framework.

config.storage.topic: The name of the Kafka topic where connector configurations are stored

Type: string — **Default**: — **Valid Values**: — **Importance**: high

---

group.id: A unique string that identifies the Connect cluster group this worker belongs to.

Type: string — **Default**: — **Valid Values**: — **Importance**: high

---

key.converter: Converter class used to convert between Kafka Connect format and the serialized form that is written to or read from Kafka, and since this is independent of connectors it allows any connector to work wi JSON and Avro.

Type: class — **Default**: — **Valid Values**: — **Importance**: high

---

offset.storage.topic: The name of the Kafka topic where connector offsets are stored

Type: string — **Default**: — **Valid Values**: — **Importance**: high

---

status.storage.topic: The name of the Kafka topic where connector and task status are stored

Type: string — **Default**: — **Valid Values**: — **Importance**: high

---

value.converter: Converter class used to convert between Kafka Connect format and the serialized form that messages written to or read from Kafka, and since this is independent of connectors it allows any connector formats include JSON and Avro.

Type: class — **Default**: — **Valid Values**: — **Importance**: high

---

bootstrap.servers: A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. T are specified here for bootstrapping—this list only impacts the initial hosts used to discover the full set of ser `host1:port1,host2:port2,...` . Since these servers are just used for the initial connection to discove this list need not contain the full set of servers (you may want more than one, though, in case a server is dow

Type: list — **Default**: localhost:9092 — **Valid Values**: — **Importance**: high

---

heartbeat.interval.ms: The expected time between heartbeats to the group coordinator when using Kafka's g the worker's session stays active and to facilitate rebalancing when new members join or leave the group. Th typically should be set no higher than 1/3 of that value. It can be adjusted even lower to control the expected

Type: int — **Default**: 3000 — **Valid Values**: — **Importance**: high

---

rebalance.timeout.ms: The maximum allowed time for each worker to join the group once a rebalance has b tasks to flush any pending data and commit offsets. If the timeout is exceeded, then the worker will be remov

Type: int — **Default**: 60000 — **Valid Values**: — **Importance**: high

---

session.timeout.ms: The timeout used to detect worker failures. The worker sends periodic heartbeats to inc broker before the expiration of this session timeout, then the broker will remove the worker from the group ar range as configured in the broker configuration by `group.min.session.timeout.ms` and `group.max`

Type: int — **Default**: 10000 — **Valid Values**: — **Importance**: high

---

ssl.key.password: The password of the private key in the key store file. This is optional for client.

Type: password — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.keystore.location: The location of the key store file. This is optional for client and can be used for two-wa

Type: string — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.keystore.password: The store password for the key store file. This is optional for client and only needed i

Type: password — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.truststore.location: The location of the trust store file.

Type: string — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.truststore.password: The password for the trust store file. If a password is not set access to the truststor

Type: password — **Default**: null — **Valid Values**: — **Importance**: high

client.dns.lookup: Controls how the client uses DNS lookups. If set to `use_all_dns_ips` then, when the be attempted to connect to before failing the connection. Applies to both bootstrap and advertised servers. I `resolve_canonical_bootstrap_servers_only` each entry will be resolved and expanded into a list

Type: string — **Default**: default — **Valid Values**: [default, use_all_dns_ips, resolve_canonical_bootstrap_s

connections.max.idle.ms: Close idle connections after the number of milliseconds specified by this config.

Type: long — **Default**: 540000 — **Valid Values**: — **Importance**: medium

connector.client.config.override.policy: Class name or alias of implementation of `ConnectorClientCon` overriden by the connector. The default implementation is `None`. The other possible policies in the framewc

Type: string — **Default**: None — **Valid Values**: — **Importance**: medium

receive.buffer.bytes: The size of the TCP receive buffer (SO_RCVBUF) to use when reading data. If the value

Type: int — **Default**: 32768 — **Valid Values**: [0,...] — **Importance**: medium

request.timeout.ms: The configuration controls the maximum amount of time the client will wait for the resp timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.

Type: int — **Default**: 40000 — **Valid Values**: [0,...] — **Importance**: medium

sasl.client.callback.handler.class: The fully qualified name of a SASL client callback handler class that imple

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium

sasl.jaas.config: JAAS login context parameters for SASL connections in the format used by JAAS configura format for the value is: '`loginModuleClass controlFlag (optionName=optionValue)*;`'. For br mechanism name in lower-case. For example, listener.name.sasl_ssl.scram-sha-256.sasl.jaas.config=com.e

Type: password — **Default**: null — **Valid Values**: — **Importance**: medium

sasl.kerberos.service.name: The Kerberos principal name that Kafka runs as. This can be defined either in K

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

sasl.login.callback.handler.class: The fully qualified name of a SASL login callback handler class that implen callback handler config must be prefixed with listener prefix and SASL mechanism name in lower-case. For e 256.sasl.login.callback.handler.class=com.example.CustomScramLoginCallbackHandler

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium

sasl.login.class: The fully qualified name of a class that implements the Login interface. For brokers, login co
name in lower-case. For example, listener.name.sasl_ssl.scram-sha-256.sasl.login.class=com.example.Custo

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.mechanism: SASL mechanism used for client connections. This may be any mechanism for which a sec

Type: string — **Default**: GSSAPI — **Valid Values**: — **Importance**: medium

---

security.protocol: Protocol used to communicate with brokers. Valid values are: PLAINTEXT, SSL, SASL_PLAI

Type: string — **Default**: PLAINTEXT — **Valid Values**: — **Importance**: medium

---

send.buffer.bytes: The size of the TCP send buffer (SO_SNDBUF) to use when sending data. If the value is -1,

Type: int — **Default**: 131072 — **Valid Values**: [0,...] — **Importance**: medium

---

ssl.enabled.protocols: The list of protocols enabled for SSL connections.

Type: list — **Default**: TLSv1.2,TLSv1.1,TLSv1 — **Valid Values**: — **Importance**: medium

---

ssl.keystore.type: The file format of the key store file. This is optional for client.

Type: string — **Default**: JKS — **Valid Values**: — **Importance**: medium

---

ssl.protocol: The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for most
TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to know

Type: string — **Default**: TLS — **Valid Values**: — **Importance**: medium

---

ssl.provider: The name of the security provider used for SSL connections. Default value is the default securit

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

ssl.truststore.type: The file format of the trust store file.

Type: string — **Default**: JKS — **Valid Values**: — **Importance**: medium

---

worker.sync.timeout.ms: When the worker is out of sync with other workers and needs to resynchronize conf
the group, and waiting a backoff period before rejoining.

Type: int — **Default**: 3000 — **Valid Values**: — **Importance**: medium

---

worker.unsync.backoff.ms: When the worker is out of sync with other workers and fails to catch up within wo
before rejoining.

Type: int — **Default**: 300000 — **Valid Values**: — **Importance**: medium

---

access.control.allow.methods: Sets the methods supported for cross origin requests by setting the Access-(
Control-Allow-Methods header allows cross origin requests for GET, POST and HEAD.

Type: string — **Default**: "" — **Valid Values**: — **Importance**: low

---

access.control.allow.origin: Value to set the Access-Control-Allow-Origin header to for REST API requests.To
application that should be permitted to access the API, or '*' to allow access from any domain. The default va

Type: string — **Default**: "" — **Valid Values**: — **Importance**: low

---

admin.listeners: List of comma-separated URIs the Admin REST API will listen on. The supported protocols a
feature. The default behavior is to use the regular listener (specified by the 'listeners' property).

Type: list — **Default**: null — **Valid Values**: org.apache.kafka.connect.runtime.WorkerConfig$AdminListen

---

client.id: An id string to pass to the server when making requests. The purpose of this is to be able to track th
application name to be included in server-side request logging.

Type: string — **Default**: "" — **Valid Values**: — **Importance**: low

---

config.providers: Comma-separated names of `ConfigProvider` classes, loaded and used in the order sp
you to replace variable references in connector configurations, such as for externalized secrets.

Type: list — **Default**: "" — **Valid Values**: — **Importance**: low

---

config.storage.replication.factor: Replication factor used when creating the configuration storage topic

Type: short — **Default**: 3 — **Valid Values**: [1,...] — **Importance**: low

---

connect.protocol: Compatibility mode for Kafka Connect Protocol

Type: string — **Default**: sessioned — **Valid Values**: [eager, compatible, sessioned] — **Importance**: low

---

header.converter: HeaderConverter class used to convert between Kafka Connect format and the serialized
values in messages written to or read from Kafka, and since this is independent of connectors it allows any c
common formats include JSON and Avro. By default, the SimpleHeaderConverter is used to serialize header

Type: class — **Default**: org.apache.kafka.connect.storage.SimpleHeaderConverter — **Valid Values**: — **In**

---

inter.worker.key.generation.algorithm: The algorithm to use for generating internal request keys

Type: string — **Default**: HmacSHA256 — **Valid Values**: Any KeyGenerator algorithm supported by the wo

---

inter.worker.key.size: The size of the key to use for signing internal requests, in bits. If null, the default key siz

Type: int — **Default**: null — **Valid Values**: — **Importance**: low

---

inter.worker.key.ttl.ms: The TTL of generated session keys used for internal request validation (in millisecon

Type: int — **Default**: 3600000 — **Valid Values**: [0,...,2147483647] — **Importance**: low

---

inter.worker.signature.algorithm: The algorithm used to sign internal requests

Type: string — **Default**: HmacSHA256 — **Valid Values**: Any MAC algorithm supported by the worker JVM

---

inter.worker.verification.algorithms: A list of permitted algorithms for verifying internal requests

Type: list — **Default**: HmacSHA256 — **Valid Values**: A list of one or more MAC algorithms, each support

---

internal.key.converter: Converter class used to convert between Kafka Connect format and the serialized for messages written to or read from Kafka, and since this is independent of connectors it allows any connector formats include JSON and Avro. This setting controls the format used for internal bookkeeping data used by use any functioning Converter implementation. Deprecated; will be removed in an upcoming version.

Type: class — **Default**: org.apache.kafka.connect.json.JsonConverter — **Valid Values**: — **Importance**: lo

---

internal.value.converter: Converter class used to convert between Kafka Connect format and the serialized f in messages written to or read from Kafka, and since this is independent of connectors it allows any connect formats include JSON and Avro. This setting controls the format used for internal bookkeeping data used by use any functioning Converter implementation. Deprecated; will be removed in an upcoming version.

Type: class — **Default**: org.apache.kafka.connect.json.JsonConverter — **Valid Values**: — **Importance**: lo

---

listeners: List of comma-separated URIs the REST API will listen on. The supported protocols are HTTP and H hostname empty to bind to default interface. Examples of legal listener lists: HTTP://myhost:8083,HTTPS://r

Type: list — **Default**: null — **Valid Values**: — **Importance**: low

---

metadata.max.age.ms: The period of time in milliseconds after which we force a refresh of metadata even if discover any new brokers or partitions.

Type: long — **Default**: 300000 — **Valid Values**: [0,...] — **Importance**: low

---

metric.reporters: A list of classes to use as metrics reporters. Implementing the `org.apache.kafka.com` classes that will be notified of new metric creation. The JmxReporter is always included to register JMX stat

Type: list — **Default**: "" — **Valid Values**: — **Importance**: low

---

metrics.num.samples: The number of samples maintained to compute metrics.

Type: int — **Default**: 2 — **Valid Values**: [1,...] — **Importance**: low

---

metrics.recording.level: The highest recording level for metrics.

Type: string — **Default**: INFO — **Valid Values**: [INFO, DEBUG] — **Importance**: low

---

metrics.sample.window.ms: The window of time a metrics sample is computed over.

Type: long — **Default**: 30000 — **Valid Values**: [0,...] — **Importance**: low

---

offset.flush.interval.ms: Interval at which to try committing offsets for tasks.

Type: long — **Default**: 60000 — **Valid Values**: — **Importance**: low

---

offset.flush.timeout.ms: Maximum number of milliseconds to wait for records to flush and partition offset d
process and restoring the offset data to be committed in a future attempt.

Type: long — **Default**: 5000 — **Valid Values**: — **Importance**: low

---

offset.storage.partitions: The number of partitions used when creating the offset storage topic

Type: int — **Default**: 25 — **Valid Values**: [1,...] — **Importance**: low

---

offset.storage.replication.factor: Replication factor used when creating the offset storage topic

Type: short — **Default**: 3 — **Valid Values**: [1,...] — **Importance**: low

---

plugin.path: List of paths separated by commas (,) that contain plugins (connectors, converters, transformat
any combination of: a) directories immediately containing jars with plugins and their dependencies b) uber-ja
containing the package directory structure of classes of plugins and their dependencies Note: symlinks will b
plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors

Type: list — **Default**: null — **Valid Values**: — **Importance**: low

---

reconnect.backoff.max.ms: The maximum amount of time in milliseconds to wait when reconnecting to a br
per host will increase exponentially for each consecutive connection failure, up to this maximum. After calcu
connection storms.

Type: long — **Default**: 1000 — **Valid Values**: [0,...] — **Importance**: low

---

reconnect.backoff.ms: The base amount of time to wait before attempting to reconnect to a given host. This
backoff applies to all connection attempts by the client to a broker.

Type: long — **Default**: 50 — **Valid Values**: [0,...] — **Importance**: low

---

rest.advertised.host.name: If this is set, this is the hostname that will be given out to other workers to conne

Type: string — **Default**: null — **Valid Values**: — **Importance**: low

---

rest.advertised.listener: Sets the advertised listener (HTTP or HTTPS) which will be given to other workers tc

Type: string — **Default**: null — **Valid Values**: — **Importance**: low

---

rest.advertised.port: If this is set, this is the port that will be given out to other workers to connect to.

Type: int — **Default**: null — **Valid Values**: — **Importance**: low

---

rest.extension.classes: Comma-separated names of `ConnectRestExtension` classes, loaded and calle `ConnectRestExtension` allows you to inject into Connect's REST API user defined resources like filters.

Type: list — **Default**: "" — **Valid Values**: — **Importance**: low

---

rest.host.name: Hostname for the REST API. If this is set, it will only bind to this interface.

Type: string — **Default**: null — **Valid Values**: — **Importance**: low

---

rest.port: Port for the REST API to listen on.

Type: int — **Default**: 8083 — **Valid Values**: — **Importance**: low

---

retry.backoff.ms: The amount of time to wait before attempting to retry a failed request to a given topic parti some failure scenarios.

Type: long — **Default**: 100 — **Valid Values**: [0,...] — **Importance**: low

---

sasl.kerberos.kinit.cmd: Kerberos kinit command path.

Type: string — **Default**: /usr/bin/kinit — **Valid Values**: — **Importance**: low

---

sasl.kerberos.min.time.before.relogin: Login thread sleep time between refresh attempts.

Type: long — **Default**: 60000 — **Valid Values**: — **Importance**: low

---

sasl.kerberos.ticket.renew.jitter: Percentage of random jitter added to the renewal time.

Type: double — **Default**: 0.05 — **Valid Values**: — **Importance**: low

---

sasl.kerberos.ticket.renew.window.factor: Login thread will sleep until the specified window factor of time fr
will try to renew the ticket.

Type: double — **Default**: 0.8 — **Valid Values**: — **Importance**: low

---

sasl.login.refresh.buffer.seconds: The amount of buffer time before credential expiration to maintain when r
occur closer to expiration than the number of buffer seconds then the refresh will be moved up to maintain a
and 3600 (1 hour); a default value of 300 (5 minutes) is used if no value is specified. This value and sasl.logir
the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

Type: short — **Default**: 300 — **Valid Values**: [0,...,3600] — **Importance**: low

---

sasl.login.refresh.min.period.seconds: The desired minimum time for the login refresh thread to wait before
and 900 (15 minutes); a default value of 60 (1 minute) is used if no value is specified. This value and sasl.log
remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

Type: short — **Default**: 60 — **Valid Values**: [0,...,900] — **Importance**: low

---

sasl.login.refresh.window.factor: Login refresh thread will sleep until the specified window factor relative to t
refresh the credential. Legal values are between 0.5 (50%) and 1.0 (100%) inclusive; a default value of 0.8 (80
OAUTHBEARER.

Type: double — **Default**: 0.8 — **Valid Values**: [0.5,...,1.0] — **Importance**: low

---

sasl.login.refresh.window.jitter: The maximum amount of random jitter relative to the credential's lifetime th
between 0 and 0.25 (25%) inclusive; a default value of 0.05 (5%) is used if no value is specified. Currently app

Type: double — **Default**: 0.05 — **Valid Values**: [0.0,...,0.25] — **Importance**: low

---

scheduled.rebalance.max.delay.ms: The maximum delay that is scheduled in order to wait for the return of o
their connectors and tasks to the group. During this period the connectors and tasks of the departed workers

Type: int — **Default**: 300000 — **Valid Values**: [0,...,2147483647] — **Importance**: low

---

ssl.cipher.suites: A list of cipher suites. This is a named combination of authentication, encryption, MAC and
a network connection using TLS or SSL network protocol. By default all the available cipher suites are suppor

Type: list — **Default**: null — **Valid Values**: — **Importance**: low

---

ssl.client.auth: Configures kafka broker to request client authentication. The following settings are common:

- `ssl.client.auth=required` If set to required client authentication is required.
- `ssl.client.auth=requested` This means client authentication is optional. unlike requested , if this
  information about itself
- `ssl.client.auth=none` This means client authentication is not needed.

Type: string — **Default**: none — **Valid Values**: — **Importance**: low

---

ssl.endpoint.identification.algorithm: The endpoint identification algorithm to validate server hostname usin

Type: string — **Default**: https — **Valid Values**: — **Importance**: low

---

ssl.keymanager.algorithm: The algorithm used by key manager factory for SSL connections. Default value is Machine.

Type: string — **Default**: SunX509 — **Valid Values**: — **Importance**: low

---

ssl.secure.random.implementation: The SecureRandom PRNG implementation to use for SSL cryptography

Type: string — **Default**: null — **Valid Values**: — **Importance**: low

---

ssl.trustmanager.algorithm: The algorithm used by trust manager factory for SSL connections. Default value Virtual Machine.

Type: string — **Default**: PKIX — **Valid Values**: — **Importance**: low

---

status.storage.partitions: The number of partitions used when creating the status storage topic

Type: int — **Default**: 5 — **Valid Values**: [1,...] — **Importance**: low

---

status.storage.replication.factor: Replication factor used when creating the status storage topic

Type: short — **Default**: 3 — **Valid Values**: [1,...] — **Importance**: low

---

task.shutdown.graceful.timeout.ms: Amount of time to wait for tasks to shutdown gracefully. This is the tota then they are waited on sequentially.

Type: long — **Default**: 5000 — **Valid Values**: — **Importance**: low

---

3.5.1 Source Connector Configs

Below is the configuration of a source connector.

name: Globally unique name to use for this connector.

Type: string — **Default**: — **Valid Values**: non-empty string without ISO control characters — **Importance**:

---

connector.class: Name or alias of the class for this connector. Must be a subclass of org.apache.kafka.conn org.apache.kafka.connect.file.FileStreamSinkConnector, you can either specify this full name, or use "FileStre bit shorter

Type: string — **Default**: — **Valid Values**: — **Importance**: high

---

tasks.max: Maximum number of tasks to use for this connector.

Type: int — **Default**: 1 — **Valid Values**: [1,...] — **Importance**: high

---

key.converter: Converter class used to convert between Kafka Connect format and the serialized form that is written to or read from Kafka, and since this is independent of connectors it allows any connector to work wi JSON and Avro.

Type: class — **Default**: null — **Valid Values**: — **Importance**: low

---

value.converter: Converter class used to convert between Kafka Connect format and the serialized form that messages written to or read from Kafka, and since this is independent of connectors it allows any connector formats include JSON and Avro.

Type: class — **Default**: null — **Valid Values**: — **Importance**: low

---

header.converter: HeaderConverter class used to convert between Kafka Connect format and the serialized values in messages written to or read from Kafka, and since this is independent of connectors it allows any c common formats include JSON and Avro. By default, the SimpleHeaderConverter is used to serialize header

Type: class — **Default**: null — **Valid Values**: — **Importance**: low

---

config.action.reload: The action that Connect should take on the connector when changes in external config properties. A value of 'none' indicates that Connect will do nothing. A value of 'restart' indicates that Connect configuration properties.The restart may actually be scheduled in the future if the external configuration prov

Type: string — **Default**: restart — **Valid Values**: [none, restart] — **Importance**: low

---

transforms: Aliases for the transformations to be applied to records.

Type: list — **Default**: "" — **Valid Values**: non-null string, unique transformation aliases — **Importance**: low

---

errors.retry.timeout: The maximum duration in milliseconds that a failed operation will be reattempted. The infinite retries.

Type: long — **Default**: 0 — **Valid Values**: — **Importance**: medium

---

errors.retry.delay.max.ms: The maximum duration in milliseconds between consecutive retry attempts. Jitte thundering herd issues.

Type: long — **Default**: 60000 — **Valid Values**: — **Importance**: medium

---

errors.tolerance: Behavior for tolerating errors during connector operation. 'none' is the default value and sigı failure; 'all' changes the behavior to skip over problematic records.

Type: string — **Default**: none — **Valid Values**: [none, all] — **Importance**: medium

---

errors.log.enable: If true, write each error and the details of the failed operation and problematic record to th errors that are not tolerated are reported.

Type: boolean — **Default**: false — **Valid Values**: — **Importance**: medium

---

errors.log.include.messages: Whether to the include in the log the Connect record that resulted in a failure. T headers from being written to log files, although some information such as topic and partition number will st

Type: boolean — **Default**: false — **Valid Values**: — **Importance**: medium

---

3.5.2 Sink Connector Configs

Below is the configuration of a sink connector.

name: Globally unique name to use for this connector.

Type: string — **Default**: — **Valid Values**: non-empty string without ISO control characters — **Importance**:

---

connector.class: Name or alias of the class for this connector. Must be a subclass of org.apache.kafka.conn org.apache.kafka.connect.file.FileStreamSinkConnector, you can either specify this full name, or use "FileStre bit shorter

Type: string — **Default**: — **Valid Values**: — **Importance**: high

---

tasks.max: Maximum number of tasks to use for this connector.

Type: int — **Default**: 1 — **Valid Values**: [1,...] — **Importance**: high

---

topics: List of topics to consume, separated by commas

Type: list — **Default**: "" — **Valid Values**: — **Importance**: high

---

topics.regex: Regular expression giving topics to consume. Under the hood, the regex is compiled to a `java` should be specified.

Type: string — **Default**: "" — **Valid Values**: valid regex — **Importance**: high

---

key.converter: Converter class used to convert between Kafka Connect format and the serialized form that is written to or read from Kafka, and since this is independent of connectors it allows any connector to work wi

JSON and Avro.

Type: class — **Default**: null — **Valid Values**: — **Importance**: low

---

value.converter: Converter class used to convert between Kafka Connect format and the serialized form that messages written to or read from Kafka, and since this is independent of connectors it allows any connector formats include JSON and Avro.

Type: class — **Default**: null — **Valid Values**: — **Importance**: low

---

header.converter: HeaderConverter class used to convert between Kafka Connect format and the serialized values in messages written to or read from Kafka, and since this is independent of connectors it allows any c common formats include JSON and Avro. By default, the SimpleHeaderConverter is used to serialize header

Type: class — **Default**: null — **Valid Values**: — **Importance**: low

---

config.action.reload: The action that Connect should take on the connector when changes in external config properties. A value of 'none' indicates that Connect will do nothing. A value of 'restart' indicates that Connect configuration properties.The restart may actually be scheduled in the future if the external configuration prov

Type: string — **Default**: restart — **Valid Values**: [none, restart] — **Importance**: low

---

transforms: Aliases for the transformations to be applied to records.

Type: list — **Default**: "" — **Valid Values**: non-null string, unique transformation aliases — **Importance**: low

---

errors.retry.timeout: The maximum duration in milliseconds that a failed operation will be reattempted. The infinite retries.

Type: long — **Default**: 0 — **Valid Values**: — **Importance**: medium

---

errors.retry.delay.max.ms: The maximum duration in milliseconds between consecutive retry attempts. Jitte thundering herd issues.

Type: long — **Default**: 60000 — **Valid Values**: — **Importance**: medium

---

errors.tolerance: Behavior for tolerating errors during connector operation. 'none' is the default value and sig failure; 'all' changes the behavior to skip over problematic records.

Type: string — **Default**: none — **Valid Values**: [none, all] — **Importance**: medium

---

errors.log.enable: If true, write each error and the details of the failed operation and problematic record to th errors that are not tolerated are reported.

Type: boolean — **Default**: false — **Valid Values**: — **Importance**: medium

---

errors.log.include.messages: Whether to the include in the log the Connect record that resulted in a failure. T
headers from being written to log files, although some information such as topic and partition number will st

Type: boolean — **Default**: false — **Valid Values**: — **Importance**: medium

---

errors.deadletterqueue.topic.name: The name of the topic to be used as the dead letter queue (DLQ) for mes
connector, or its transformations or converters. The topic name is blank by default, which means that no mes

Type: string — **Default**: "" — **Valid Values**: — **Importance**: medium

---

errors.deadletterqueue.topic.replication.factor: Replication factor used to create the dead letter queue topic

Type: short — **Default**: 3 — **Valid Values**: — **Importance**: medium

---

errors.deadletterqueue.context.headers.enable: If true, add headers containing error context to the message
from the original record, all error context header keys, all error context header keys will start with `__connec`

Type: boolean — **Default**: false — **Valid Values**: — **Importance**: medium

---

3.6 Kafka Streams Configs

Below is the configuration of the Kafka Streams client library.

application.id: An identifier for the stream processing application. Must be unique within the Kafka cluster. It
membership management, 3) the changelog topic prefix.

Type: string — **Default**: — **Valid Values**: — **Importance**: high

---

bootstrap.servers: A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. T
are specified here for bootstrapping—this list only impacts the initial hosts used to discover the full set of ser
`host1:port1,host2:port2,...` . Since these servers are just used for the initial connection to discove
this list need not contain the full set of servers (you may want more than one, though, in case a server is dow

Type: list — **Default**: — **Valid Values**: — **Importance**: high

---

replication.factor: The replication factor for change log topics and repartition topics created by the stream pr

Type: int — **Default**: 1 — **Valid Values**: — **Importance**: high

---

state.dir: Directory location for state store. This path must be unique for each streams instance sharing the s

Type: string — **Default**: /tmp/kafka-streams — **Valid Values**: — **Importance**: high

---

cache.max.bytes.buffering: Maximum number of memory bytes to be used for buffering across all threads

Type: long — **Default**: 10485760 — **Valid Values**: [0,...] — **Importance**: medium

---

client.id: An ID prefix string used for the client IDs of internal consumer, producer and restore-consumer, with

Type: string — **Default**: "" — **Valid Values**: — **Importance**: medium

---

default.deserialization.exception.handler: Exception handling class that implements the `org.apache.kafka.streams.errors.DeserializationExceptionHandler` interface.

Type: class — **Default**: org.apache.kafka.streams.errors.LogAndFailExceptionHandler — **Valid Values**: -

---

default.key.serde: Default serializer / deserializer class for key that implements the `org.apache.kafka.(` windowed serde class is used, one needs to set the inner serde class that implements the `org.apache.ka` 'default.windowed.key.serde.inner' or 'default.windowed.value.serde.inner' as well

Type: class — **Default**: org.apache.kafka.common.serialization.Serdes$ByteArraySerde — **Valid Values**:

---

default.production.exception.handler: Exception handling class that implements the `org.apache.kafka` interface.

Type: class — **Default**: org.apache.kafka.streams.errors.DefaultProductionExceptionHandler — **Valid Va**

---

default.timestamp.extractor: Default timestamp extractor class that implements the `org.apache.kafka`

Type: class — **Default**: org.apache.kafka.streams.processor.FailOnInvalidTimestamp — **Valid Values**: —

---

default.value.serde: Default serializer / deserializer class for value that implements the `org.apache.kafl` windowed serde class is used, one needs to set the inner serde class that implements the `org.apache.ka` 'default.windowed.key.serde.inner' or 'default.windowed.value.serde.inner' as well

Type: class — **Default**: org.apache.kafka.common.serialization.Serdes$ByteArraySerde — **Valid Values**:

---

max.task.idle.ms: Maximum amount of time a stream task will stay idle when not all of its partition buffers c across multiple input streams.

Type: long — **Default**: 0 — **Valid Values**: — **Importance**: medium

---

num.standby.replicas: The number of standby replicas for each task.

Type: int — **Default**: 0 — **Valid Values**: — **Importance**: medium

---

num.stream.threads: The number of threads to execute stream processing.

Type: int — **Default**: 1 — **Valid Values**: — **Importance**: medium

---

processing.guarantee: The processing guarantee that should be used. Possible values are `at_least_onc` processing requires a cluster of at least three brokers by default what is the recommended setting for produc setting `transaction.state.log.replication.factor` and `transaction.state.log.min.is`

Type: string — **Default**: at_least_once — **Valid Values**: [at_least_once, exactly_once] — **Importance**: mec

---

security.protocol: Protocol used to communicate with brokers. Valid values are: PLAINTEXT, SSL, SASL_PLAI

Type: string — **Default**: PLAINTEXT — **Valid Values**: — **Importance**: medium

---

topology.optimization: A configuration telling Kafka Streams if it should optimize the topology, disabled by de

Type: string — **Default**: none — **Valid Values**: [none, all] — **Importance**: medium

---

application.server: A host:port pair pointing to an embedded user defined endpoint that can be used for disc application

Type: string — **Default**: "" — **Valid Values**: — **Importance**: low

---

buffered.records.per.partition: Maximum number of records to buffer per partition.

Type: int — **Default**: 1000 — **Valid Values**: — **Importance**: low

---

commit.interval.ms: The frequency with which to save the position of the processor. (Note, if `processing` `100` , otherwise the default value is `30000` .

Type: long — **Default**: 30000 — **Valid Values**: [0,...] — **Importance**: low

---

connections.max.idle.ms: Close idle connections after the number of milliseconds specified by this config.

Type: long — **Default**: 540000 — **Valid Values**: — **Importance**: low

---

metadata.max.age.ms: The period of time in milliseconds after which we force a refresh of metadata even if discover any new brokers or partitions.

Type: long — **Default**: 300000 — **Valid Values**: [0,...] — **Importance**: low

---

metric.reporters: A list of classes to use as metrics reporters. Implementing the `org.apache.kafka.com` classes that will be notified of new metric creation. The JmxReporter is always included to register JMX stati

Type: list — **Default**: "" — **Valid Values**: — **Importance**: low

---

metrics.num.samples: The number of samples maintained to compute metrics.

Type: int — **Default**: 2 — **Valid Values**: [1,...] — **Importance**: low

---

metrics.recording.level: The highest recording level for metrics.

Type: string — **Default**: INFO — **Valid Values**: [INFO, DEBUG] — **Importance**: low

---

metrics.sample.window.ms: The window of time a metrics sample is computed over.

Type: long — **Default**: 30000 — **Valid Values**: [0,...] — **Importance**: low

---

partition.grouper: Partition grouper class that implements the `org.apache.kafka.streams.processo`
deprecated and will be removed in 3.0.0 release.

Type: class — **Default**: org.apache.kafka.streams.processor.DefaultPartitionGrouper — **Valid Values**: —

---

poll.ms: The amount of time in milliseconds to block waiting for input.

Type: long — **Default**: 100 — **Valid Values**: — **Importance**: low

---

receive.buffer.bytes: The size of the TCP receive buffer (SO_RCVBUF) to use when reading data. If the value

Type: int — **Default**: 32768 — **Valid Values**: [-1,...] — **Importance**: low

---

reconnect.backoff.max.ms: The maximum amount of time in milliseconds to wait when reconnecting to a br
per host will increase exponentially for each consecutive connection failure, up to this maximum. After calcu
connection storms.

Type: long — **Default**: 1000 — **Valid Values**: [0,...] — **Importance**: low

---

reconnect.backoff.ms: The base amount of time to wait before attempting to reconnect to a given host. This
backoff applies to all connection attempts by the client to a broker.

Type: long — **Default**: 50 — **Valid Values**: [0,...] — **Importance**: low

---

request.timeout.ms: The configuration controls the maximum amount of time the client will wait for the resp
timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.

Type: int — **Default**: 40000 — **Valid Values**: [0,...] — **Importance**: low

---

retries: Setting a value greater than zero will cause the client to resend any request that fails with a potentiall

Type: int — **Default**: 0 — **Valid Values**: [0,...,2147483647] — **Importance**: low

---

retry.backoff.ms: The amount of time to wait before attempting to retry a failed request to a given topic parti
some failure scenarios.

Type: long — **Default**: 100 — **Valid Values**: [0,...] — **Importance**: low

---

rocksdb.config.setter: A Rocks DB config setter class or class name that implements the `org.apache.ka`

Type: class — **Default**: null — **Valid Values**: — **Importance**: low

---

send.buffer.bytes: The size of the TCP send buffer (SO_SNDBUF) to use when sending data. If the value is -1,

Type: int — **Default**: 131072 — **Valid Values**: [-1,...] — **Importance**: low

---

state.cleanup.delay.ms: The amount of time in milliseconds to wait before deleting state when a partition ha at least `state.cleanup.delay.ms` will be removed

Type: long — **Default**: 600000 — **Valid Values**: — **Importance**: low

---

upgrade.from: Allows upgrading in a backward compatible way. This is needed when upgrading from [0.10.0, upgrading from 2.4 to a newer version it is not required to specify this config. Default is null. Accepted values "2.3" (for upgrading from the corresponding old version).

Type: string — **Default**: null — **Valid Values**: [null, 0.10.0, 0.10.1, 0.10.2, 0.11.0, 1.0, 1.1, 2.0, 2.1, 2.2, 2.3]

---

windowstore.changelog.additional.retention.ms: Added to a windows maintainMs to ensure data is not dele

Type: long — **Default**: 86400000 — **Valid Values**: — **Importance**: low

---

3.7 Admin Configs

Below is the configuration of the Kafka Admin client library.

bootstrap.servers: A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. T are specified here for bootstrapping—this list only impacts the initial hosts used to discover the full set of ser `host1:port1,host2:port2,...` . Since these servers are just used for the initial connection to discove this list need not contain the full set of servers (you may want more than one, though, in case a server is dow

Type: list — **Default**: — **Valid Values**: — **Importance**: high

---

ssl.key.password: The password of the private key in the key store file. This is optional for client.

Type: password — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.keystore.location: The location of the key store file. This is optional for client and can be used for two-wa

Type: string — **Default**: null — **Valid Values**: — **Importance**: high

---

ssl.keystore.password: The store password for the key store file. This is optional for client and only needed i

Type: password — **Default**: null — **Valid Values**: — **Importance**: high

ssl.truststore.location: The location of the trust store file.

Type: string — **Default**: null — **Valid Values**: — **Importance**: high

ssl.truststore.password: The password for the trust store file. If a password is not set access to the truststor

Type: password — **Default**: null — **Valid Values**: — **Importance**: high

client.dns.lookup: Controls how the client uses DNS lookups. If set to `use_all_dns_ips` then, when the be attempted to connect to before failing the connection. Applies to both bootstrap and advertised servers. I `resolve_canonical_bootstrap_servers_only` each entry will be resolved and expanded into a list

Type: string — **Default**: default — **Valid Values**: [default, use_all_dns_ips, resolve_canonical_bootstrap_s

client.id: An id string to pass to the server when making requests. The purpose of this is to be able to track th application name to be included in server-side request logging.

Type: string — **Default**: "" — **Valid Values**: — **Importance**: medium

connections.max.idle.ms: Close idle connections after the number of milliseconds specified by this config.

Type: long — **Default**: 300000 — **Valid Values**: — **Importance**: medium

receive.buffer.bytes: The size of the TCP receive buffer (SO_RCVBUF) to use when reading data. If the value

Type: int — **Default**: 65536 — **Valid Values**: [-1,...] — **Importance**: medium

request.timeout.ms: The configuration controls the maximum amount of time the client will wait for the resp timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.

Type: int — **Default**: 120000 — **Valid Values**: [0,...] — **Importance**: medium

sasl.client.callback.handler.class: The fully qualified name of a SASL client callback handler class that imple

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium

sasl.jaas.config: JAAS login context parameters for SASL connections in the format used by JAAS configura format for the value is: '`loginModuleClass controlFlag (optionName=optionValue)*;`'. For br mechanism name in lower-case. For example, listener.name.sasl_ssl.scram-sha-256.sasl.jaas.config=com.e:

Type: password — **Default**: null — **Valid Values**: — **Importance**: medium

sasl.kerberos.service.name: The Kerberos principal name that Kafka runs as. This can be defined either in K

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

sasl.login.callback.handler.class: The fully qualified name of a SASL login callback handler class that implen
callback handler config must be prefixed with listener prefix and SASL mechanism name in lower-case. For e
256.sasl.login.callback.handler.class=com.example.CustomScramLoginCallbackHandler

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.login.class: The fully qualified name of a class that implements the Login interface. For brokers, login cc
name in lower-case. For example, listener.name.sasl_ssl.scram-sha-256.sasl.login.class=com.example.Custo

Type: class — **Default**: null — **Valid Values**: — **Importance**: medium

---

sasl.mechanism: SASL mechanism used for client connections. This may be any mechanism for which a sec

Type: string — **Default**: GSSAPI — **Valid Values**: — **Importance**: medium

---

security.protocol: Protocol used to communicate with brokers. Valid values are: PLAINTEXT, SSL, SASL_PLAI

Type: string — **Default**: PLAINTEXT — **Valid Values**: — **Importance**: medium

---

send.buffer.bytes: The size of the TCP send buffer (SO_SNDBUF) to use when sending data. If the value is -1

Type: int — **Default**: 131072 — **Valid Values**: [-1,...] — **Importance**: medium

---

ssl.enabled.protocols: The list of protocols enabled for SSL connections.

Type: list — **Default**: TLSv1.2,TLSv1.1,TLSv1 — **Valid Values**: — **Importance**: medium

---

ssl.keystore.type: The file format of the key store file. This is optional for client.

Type: string — **Default**: JKS — **Valid Values**: — **Importance**: medium

---

ssl.protocol: The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for most
TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to know

Type: string — **Default**: TLS — **Valid Values**: — **Importance**: medium

---

ssl.provider: The name of the security provider used for SSL connections. Default value is the default securit

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

ssl.truststore.type: The file format of the trust store file.

Type: string — **Default**: JKS — **Valid Values**: — **Importance**: medium

---

metadata.max.age.ms: The period of time in milliseconds after which we force a refresh of metadata even if discover any new brokers or partitions.

Type: long — **Default**: 300000 — **Valid Values**: [0,...] — **Importance**: low

---

metric.reporters: A list of classes to use as metrics reporters. Implementing the `org.apache.kafka.com` classes that will be notified of new metric creation. The JmxReporter is always included to register JMX stati

Type: list — **Default**: "" — **Valid Values**: — **Importance**: low

---

metrics.num.samples: The number of samples maintained to compute metrics.

Type: int — **Default**: 2 — **Valid Values**: [1,...] — **Importance**: low

---

metrics.recording.level: The highest recording level for metrics.

Type: string — **Default**: INFO — **Valid Values**: [INFO, DEBUG] — **Importance**: low

---

metrics.sample.window.ms: The window of time a metrics sample is computed over.

Type: long — **Default**: 30000 — **Valid Values**: [0,...] — **Importance**: low

---

reconnect.backoff.max.ms: The maximum amount of time in milliseconds to wait when reconnecting to a br per host will increase exponentially for each consecutive connection failure, up to this maximum. After calcu connection storms.

Type: long — **Default**: 1000 — **Valid Values**: [0,...] — **Importance**: low

---

reconnect.backoff.ms: The base amount of time to wait before attempting to reconnect to a given host. This backoff applies to all connection attempts by the client to a broker.

Type: long — **Default**: 50 — **Valid Values**: [0,...] — **Importance**: low

---

retries: Setting a value greater than zero will cause the client to resend any request that fails with a potentiall

Type: int — **Default**: 5 — **Valid Values**: [0,...] — **Importance**: low

---

retry.backoff.ms: The amount of time to wait before attempting to retry a failed request. This avoids repeate

Type: long — **Default**: 100 — **Valid Values**: [0,...] — **Importance**: low

---

sasl.kerberos.kinit.cmd: Kerberos kinit command path.

Type: string — **Default**: /usr/bin/kinit — **Valid Values**: — **Importance**: low

---

sasl.kerberos.min.time.before.relogin: Login thread sleep time between refresh attempts.

Type: long — **Default**: 60000 — **Valid Values**: — **Importance**: low

---

sasl.kerberos.ticket.renew.jitter: Percentage of random jitter added to the renewal time.

Type: double — **Default**: 0.05 — **Valid Values**: — **Importance**: low

---

sasl.kerberos.ticket.renew.window.factor: Login thread will sleep until the specified window factor of time fr will try to renew the ticket.

Type: double — **Default**: 0.8 — **Valid Values**: — **Importance**: low

---

sasl.login.refresh.buffer.seconds: The amount of buffer time before credential expiration to maintain when r occur closer to expiration than the number of buffer seconds then the refresh will be moved up to maintain a and 3600 (1 hour); a default value of 300 (5 minutes) is used if no value is specified. This value and sasl.logi the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

Type: short — **Default**: 300 — **Valid Values**: [0,...,3600] — **Importance**: low

---

sasl.login.refresh.min.period.seconds: The desired minimum time for the login refresh thread to wait before and 900 (15 minutes); a default value of 60 (1 minute) is used if no value is specified. This value and sasl.log remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

Type: short — **Default**: 60 — **Valid Values**: [0,...,900] — **Importance**: low

---

sasl.login.refresh.window.factor: Login refresh thread will sleep until the specified window factor relative to refresh the credential. Legal values are between 0.5 (50%) and 1.0 (100%) inclusive; a default value of 0.8 (8C OAUTHBEARER.

Type: double — **Default**: 0.8 — **Valid Values**: [0.5,...,1.0] — **Importance**: low

---

sasl.login.refresh.window.jitter: The maximum amount of random jitter relative to the credential's lifetime th between 0 and 0.25 (25%) inclusive; a default value of 0.05 (5%) is used if no value is specified. Currently app

Type: double — **Default**: 0.05 — **Valid Values**: [0.0,...,0.25] — **Importance**: low

---

security.providers: A list of configurable creator classes each returning a provider implementing security alg `org.apache.kafka.common.security.auth.SecurityProviderCreator` interface.

Type: string — **Default**: null — **Valid Values**: — **Importance**: low

---

ssl.cipher.suites: A list of cipher suites. This is a named combination of authentication, encryption, MAC and a network connection using TLS or SSL network protocol. By default all the available cipher suites are suppo

Type: list — **Default**: null — **Valid Values**: — **Importance**: low

---

ssl.endpoint.identification.algorithm: The endpoint identification algorithm to validate server hostname usin

Type: string — **Default**: https — **Valid Values**: — **Importance**: low

---

ssl.keymanager.algorithm: The algorithm used by key manager factory for SSL connections. Default value is Machine.

Type: string — **Default**: SunX509 — **Valid Values**: — **Importance**: low

---

ssl.secure.random.implementation: The SecureRandom PRNG implementation to use for SSL cryptography

Type: string — **Default**: null — **Valid Values**: — **Importance**: low

---

ssl.trustmanager.algorithm: The algorithm used by trust manager factory for SSL connections. Default value Virtual Machine.

Type: string — **Default**: PKIX — **Valid Values**: — **Importance**: low

---

4. DESIGN

4.1 Motivation

We designed Kafka to be able to act as a unified platform for handling all the real-time data feeds a large company of use cases.

It would have to have high-throughput to support high volume event streams such as real-time log aggregation.

It would need to deal gracefully with large data backlogs to be able to support periodic data loads from offline sys

It also meant the system would have to handle low-latency delivery to handle more traditional messaging use-case

We wanted to support partitioned, distributed, real-time processing of these feeds to create new, derived feeds. Th

Finally in cases where the stream is fed into other data systems for serving, we knew the system would have to be failures.

Supporting these uses led us to a design with a number of unique elements, more akin to a database log than a tra design in the following sections.

4.2 Persistence

Don't fear the filesystem!

Kafka relies heavily on the filesystem for storing and caching messages. There is a general perception that "disks a
can offer competitive performance. In fact disks are both much slower and much faster than people expect depen
can often be as fast as the network.

The key fact about disk performance is that the throughput of hard drives has been diverging from the latency of a
writes on a [JBOD](#) configuration with six 7200rpm SATA RAID-5 array is about 600MB/sec but the performance of r
These linear reads and writes are the most predictable of all usage patterns, and are heavily optimized by the oper
write-behind techniques that prefetch data in large block multiples and group smaller logical writes into large phys
[ACM Queue article](#); they actually find that [sequential disk access can in some cases be faster than random memo](#)

To compensate for this performance divergence, modern operating systems have become increasingly aggressive
happily divert *all* free memory to disk caching with little performance penalty when the memory is reclaimed. All di
cannot easily be turned off without using direct I/O, so even if a process maintains an in-process cache of the data
storing everything twice.

Furthermore, we are building on top of the JVM, and anyone who has spent any time with Java memory usage kno

1. The memory overhead of objects is very high, often doubling the size of the data stored (or worse).
2. Java garbage collection becomes increasingly fiddly and slow as the in-heap data increases.

As a result of these factors using the filesystem and relying on pagecache is superior to maintaining an in-memory
by having automatic access to all free memory, and likely double again by storing a compact byte structure rather
30GB on a 32GB machine without GC penalties. Furthermore, this cache will stay warm even if the service is restar
memory (which for a 10GB cache may take 10 minutes) or else it will need to start with a completely cold cache (w
simplifies the code as all logic for maintaining coherency between the cache and filesystem is now in the OS, whic
in-process attempts. If your disk usage favors linear reads then read-ahead is effectively pre-populating this cache

This suggests a design which is very simple: rather than maintain as much as possible in-memory and flush it all c
that. All data is immediately written to a persistent log on the filesystem without necessarily flushing to disk. In eff
pagecache.

This style of pagecache-centric design is described in an [article](#) on the design of Varnish here (along with a health

Constant Time Suffices

The persistent data structure used in messaging systems are often a per-consumer queue with an associated BTr
maintain metadata about messages. BTrees are the most versatile data structure available, and make it possible t
semantics in the messaging system. They do come with a fairly high cost, though: Btree operations are O(log N). N
time, but this is not true for disk operations. Disk seeks come at 10 ms a pop, and each disk can do only one seek
seeks leads to very high overhead. Since storage systems mix very fast cached operations with very slow physical
often superlinear as data increases with fixed cache--i.e. doubling your data makes things much worse than twice

Intuitively a persistent queue could be built on simple reads and appends to files as is commonly the case with log
are O(1) and reads do not block writes or each other. This has obvious performance advantages since the perform
now take full advantage of a number of cheap, low-rotational speed 1+TB SATA drives. Though they have poor see
large reads and writes and come at 1/3 the price and 3x the capacity.

Having access to virtually unlimited disk space without any performance penalty means that we can provide some
Kafka, instead of attempting to delete messages as soon as they are consumed, we can retain messages for a rel
flexibility for consumers, as we will describe.

4.3 Efficiency

We have put significant effort into efficiency. One of our primary use cases is handling web activity data, which is v
Furthermore, we assume each message published is read by at least one consumer (often many), hence we strive

We have also found, from experience building and running a number of similar systems, that efficiency is a key to e
service can easily become a bottleneck due to a small bump in usage by the application, such small changes will c
application will tip-over under load before the infrastructure. This is particularly important when trying to run a cen
on a centralized cluster as changes in usage patterns are a near-daily occurrence.

We discussed disk efficiency in the previous section. Once poor disk access patterns have been eliminated, there a
many small I/O operations, and excessive byte copying.

The small I/O problem happens both between the client and the server and in the server's own persistent operatio

To avoid this, our protocol is built around a "message set" abstraction that naturally groups messages together. Th
amortize the overhead of the network roundtrip rather than sending a single message at a time. The server in turn
consumer fetches large linear chunks at a time.

This simple optimization produces orders of magnitude speed up. Batching leads to larger network packets, large
all of which allows Kafka to turn a bursty stream of random message writes into linear writes that flow to the cons

The other inefficiency is in byte copying. At low message rates this is not an issue, but under load the impact is sig
format that is shared by the producer, the broker, and the consumer (so data chunks can be transferred without mo

The message log maintained by the broker is itself just a directory of files, each populated by a sequence of mess
the producer and consumer. Maintaining this common format allows optimization of the most important operatio
systems offer a highly optimized code path for transferring data out of pagecache to a socket; in Linux this is done

To understand the impact of sendfile, it is important to understand the common data path for transfer of data fron

1. The operating system reads data from the disk into pagecache in kernel space
2. The application reads the data from kernel space into a user-space buffer
3. The application writes the data back into kernel space into a socket buffer
4. The operating system copies the data from the socket buffer to the NIC buffer where it is sent over the netwo

This is clearly inefficient, there are four copies and two system calls. Using sendfile, this re-copying is avoided by a
directly. So in this optimized path, only the final copy to the NIC buffer is needed.

We expect a common use case to be multiple consumers on a topic. Using the zero-copy optimization above, data
consumption instead of being stored in memory and copied out to user-space every time it is read. This allows me
network connection.

This combination of pagecache and sendfile means that on a Kafka cluster where the consumers are mostly caug
will be serving data entirely from cache.

For more background on the sendfile and zero-copy support in Java, see this article.

End-to-end Batch Compression

In some cases the bottleneck is actually not CPU or disk but network bandwidth. This is particularly true for a data
a wide-area network. Of course, the user can always compress its messages one at a time without any support ne
as much of the redundancy is due to repetition between messages of the same type (e.g. field names in JSON or u
compression requires compressing multiple messages together rather than compressing each message individua

Kafka supports this with an efficient batching format. A batch of messages can be clumped together compressed
written in compressed form and will remain compressed in the log and will only be decompressed by the consume

Kafka supports GZIP, Snappy, LZ4 and ZStandard compression protocols. More details on compression can be fou

4.4 The Producer

Load balancing

The producer sends data directly to the broker that is the leader for the partition without any intervening routing tie
request for metadata about which servers are alive and where the leaders for the partitions of a topic are at any gi

The client controls which partition it publishes messages to. This can be done at random, implementing a kind of r
partitioning function. We expose the interface for semantic partitioning by allowing the user to specify a key to par
option to override the partition function if need be). For example if the key chosen was a user id then all data for a
allow consumers to make locality assumptions about their consumption. This style of partitioning is explicitly des

Asynchronous send

Batching is one of the big drivers of efficiency, and to enable batching the Kafka producer will attempt to accumula
request. The batching can be configured to accumulate no more than a fixed number of messages and to wait no
allows the accumulation of more bytes to send, and few larger I/O operations on the servers. This buffering is con
additional latency for better throughput.

Details on configuration and the api for the producer can be found elsewhere in the documentation.

4.5 The Consumer

The Kafka consumer works by issuing "fetch" requests to the brokers leading the partitions it wants to consume. T
receives back a chunk of log beginning from that position. The consumer thus has significant control over this pos

Push vs. pull

An initial question we considered is whether consumers should pull data from brokers or brokers should push data design, shared by most messaging systems, where data is pushed to the broker from the producer and pulled from as [Scribe](#) and [Apache Flume](#), follow a very different push-based path where data is pushed downstream. There are system has difficulty dealing with diverse consumers as the broker controls the rate at which data is transferred. T maximum possible rate; unfortunately, in a push system this means the consumer tends to be overwhelmed when of service attack, in essence). A pull-based system has the nicer property that the consumer simply falls behind ar backoff protocol by which the consumer can indicate it is overwhelmed, but getting the rate of transfer to fully utili Previous attempts at building systems in this fashion led us to go with a more traditional pull model.

Another advantage of a pull-based system is that it lends itself to aggressive batching of data sent to the consume immediately or accumulate more data and then send it later without knowledge of whether the downstream consu this will result in sending a single message at a time only for the transfer to end up being buffered anyway, which i pulls all available messages after its current position in the log (or up to some configurable max size). So one gets

The deficiency of a naive pull-based system is that if the broker has no data the consumer may end up polling in a we have parameters in our pull request that allow the consumer request to block in a "long poll" waiting until data a available to ensure large transfer sizes).

You could imagine other possible designs which would be only pull, end-to-end. The producer would locally write to pulling from them. A similar type of "store-and-forward" producer is often proposed. This is intriguing but we felt no producers. Our experience running persistent data systems at scale led us to feel that involving thousands of disks things more reliable and would be a nightmare to operate. And in practice we have found that we can run a pipeline persistence.

Consumer Position

Keeping track of *what* has been consumed is, surprisingly, one of the key performance points of a messaging syst

Most messaging systems keep metadata about what messages have been consumed on the broker. That is, as a that fact locally immediately or it may wait for acknowledgement from the consumer. This is a fairly intuitive choic this state could go. Since the data structures used for storage in many messaging systems scale poorly, this is als can immediately delete it, keeping the data size small.

What is perhaps not obvious is that getting the broker and consumer to come into agreement about what has been message as **consumed** immediately every time it is handed out over the network, then if the consumer fails to pro or whatever) that message will be lost. To solve this problem, many messaging systems add an acknowledgemen **consumed** when they are sent; the broker waits for a specific acknowledgement from the consumer to record the r messages, but creates new problems. First of all, if the consumer processes the message but fails before it can se twice. The second problem is around performance, now the broker must keep multiple states about every single m then to mark it as permanently consumed so that it can be removed). Tricky problems must be dealt with, like wha

Kafka handles this differently. Our topic is divided into a set of totally ordered partitions, each of which is consume group at any given time. This means that the position of a consumer in each partition is just a single integer, the of what has been consumed very small, just one number for each partition. This state can be periodically checkpoint cheap.

There is a side benefit of this decision. A consumer can deliberately *rewind* back to an old offset and re-consume d
to be an essential feature for many consumers. For example, if the consumer code has a bug and is discovered af
those messages once the bug is fixed.

Offline Data Load

Scalable persistence allows for the possibility of consumers that only periodically consume such as batch data loa
Hadoop or a relational data warehouse.

In the case of Hadoop we parallelize the data load by splitting the load over individual map tasks, one for each noc
loading. Hadoop provides the task management, and tasks which fail can restart without danger of duplicate data

Static Membership

Static membership aims to improve the availability of stream applications, consumer groups and other applicatior
protocol relies on the group coordinator to allocate entity ids to group members. These generated ids are ephemer
based apps, this "dynamic membership" can cause a large percentage of tasks re-assigned to different instances d
configuration updates and periodic restarts. For large state applications, shuffled tasks need a long time to recove
partially or entirely unavailable. Motivated by this observation, Kafka's group management protocol allows group n
remains unchanged based on those ids, thus no rebalance will be triggered.

If you want to use static membership,

- Upgrade both broker cluster and client apps to 2.3 or beyond, and also make sure the upgraded brokers are usi
well.
- Set the config `ConsumerConfig#GROUP_INSTANCE_ID_CONFIG` to a unique value for each consumer ins
- For Kafka Streams applications, it is sufficient to set a unique `ConsumerConfig#GROUP_INSTANCE_ID_CO`
used threads for an instance.

If your broker is on an older version than 2.3, but you choose to set `ConsumerConfig#GROUP_INSTANCE_ID_(`
version and then throws an UnsupportedException. If you accidentally configure duplicate ids for different instanc
client to shutdown immediately by triggering a `org.apache.kafka.common.errors.FencedInstanceIdE`

4.6 Message Delivery Semantics

Now that we understand a little about how producers and consumers work, let's discuss the semantic guarantees
multiple possible message delivery guarantees that could be provided:

- *At most once*—Messages may be lost but are never redelivered.
- *At least once*—Messages are never lost but may be redelivered.
- *Exactly once*—this is what people actually want, each message is delivered once and only once.

It's worth noting that this breaks down into two problems: the durability guarantees for publishing a message and t

Many systems claim to provide "exactly once" delivery semantics, but it is important to read the fine print, most of
where consumers or producers can fail, cases where there are multiple consumer processes, or cases where data

Kafka's semantics are straight-forward. When publishing a message we have a notion of the message being "comi not be lost as long as one broker that replicates the partition to which this message was written remains "alive". Tl description of which types of failures we attempt to handle will be described in more detail in the next section. For the guarantees to the producer and consumer. If a producer attempts to publish a message and experiences a net the message was committed. This is similar to the semantics of inserting into a database table with an autogener

Prior to 0.11.0.0, if a producer failed to receive a response indicating that a message was committed, it had little c delivery semantics since the message may be written to the log again during resending if the original request had supports an idempotent delivery option which guarantees that resending will not result in duplicate entries in the l deduplicates messages using a sequence number that is sent by the producer along with every message. Also be messages to multiple topic partitions using transaction-like semantics: i.e. either all messages are successfully w once processing between Kafka topics (described below).

Not all use cases require such strong guarantees. For uses which are latency sensitive we allow the producer to sp wants to wait on the message being committed this can take on the order of 10 ms. However the producer can als asynchronously or that it wants to wait only until the leader (but not necessarily the followers) have the message.

Now let's describe the semantics from the point-of-view of the consumer. All replicas have the exact same log witl the consumer never crashed it could just store this position in memory, but if the consumer fails and we want this process will need to choose an appropriate position from which to start processing. Let's say the consumer reads messages and updating its position.

1. It can read the messages, then save its position in the log, and finally process the messages. In this case ther its position but before saving the output of its message processing. In this case the process that took over pr messages prior to that position had not been processed. This corresponds to "at-most-once" semantics as in
2. It can read the messages, process the messages, and finally save its position. In this case there is a possibilit but before saving its position. In this case when the new process takes over the first few messages it receives least-once" semantics in the case of consumer failure. In many cases messages have a primary key and so tl overwrites a record with another copy of itself).

So what about exactly once semantics (i.e. the thing you actually want)? When consuming from a Kafka topic and can leverage the new transactional producer capabilities in 0.11.0.0 that were mentioned above. The consumer's p offset to Kafka in the same transaction as the output topics receiving the processed data. If the transaction is abo produced data on the output topics will not be visible to other consumers, depending on their "isolation level." In th visible to consumers even if they were part of an aborted transaction, but in "read_committed," the consumer will c (and any messages which were not part of a transaction).

When writing to an external system, the limitation is in the need to coordinate the consumer's position with what is be to introduce a two-phase commit between the storage of the consumer position and the storage of the consum letting the consumer store its offset in the same place as its output. This is better because many of the output sys phase commit. As an example of this, consider a Kafka Connect connector which populates data in HDFS along w either data and offsets are both updated or neither is. We follow similar patterns for many other data systems whi do not have a primary key to allow for deduplication.

So effectively Kafka supports exactly-once delivery in Kafka Streams, and the transactional producer/consumer ca transferring and processing data between Kafka topics. Exactly-once delivery for other destination systems genera offset which makes implementing this feasible (see also Kafka Connect). Otherwise, Kafka guarantees at-least-on once delivery by disabling retries on the producer and committing offsets in the consumer prior to processing a ba

4.7 Replication

Kafka replicates the log for each topic's partitions across a configurable number of servers (you can set this replic failover to these replicas when a server in the cluster fails so messages remain available in the presence of failure

Other messaging systems provide some replication-related features, but, in our (totally biased) opinion, this appea downsides: replicas are inactive, throughput is heavily impacted, it requires fiddly manual configuration, etc. Kafka implement un-replicated topics as replicated topics where the replication factor is one.

The unit of replication is the topic partition. Under non-failure conditions, each partition in Kafka has a single leade including the leader constitute the replication factor. All reads and writes go to the leader of the partition. Typically evenly distributed among brokers. The logs on the followers are identical to the leader's log—all have the same off given time the leader may have a few as-yet unreplicated messages at the end of its log).

Followers consume messages from the leader just as a normal Kafka consumer would and apply them to their ow property of allowing the follower to naturally batch together log entries they are applying to their log.

As with most distributed systems automatically handling failures requires having a precise definition of what it me conditions

1. A node must be able to maintain its session with ZooKeeper (via ZooKeeper's heartbeat mechanism)
2. If it is a follower it must replicate the writes happening on the leader and not fall "too far" behind

We refer to nodes satisfying these two conditions as being "in sync" to avoid the vagueness of "alive" or "failed". Th dies, gets stuck, or falls behind, the leader will remove it from the list of in sync replicas. The determination of stuc configuration.

In distributed systems terminology we only attempt to handle a "fail/recover" model of failures where nodes sudde knowing that they have died). Kafka does not handle so-called "Byzantine" failures in which nodes produce arbitrar

We can now more precisely define that a message is considered committed when all in sync replicas for that parti given out to the consumer. This means that the consumer need not worry about potentially seeing a message that the option of either waiting for the message to be committed or not, depending on their preference for tradeoff bet acks setting that the producer uses. Note that topics have a setting for the "minimum number" of in-sync replicas t a message has been written to the full set of in-sync replicas. If a less stringent acknowledgement is requested by consumed, even if the number of in-sync replicas is lower than the minimum (e.g. it can be as low as just the leade

The guarantee that Kafka offers is that a committed message will not be lost, as long as there is at least one in syı

Kafka will remain available in the presence of node failures after a short fail-over period, but may not remain availa

Replicated Logs: Quorums, ISRs, and State Machines (Oh my!)

At its heart a Kafka partition is a replicated log. The replicated log is one of the most basic primitives in distributec one. A replicated log can be used by other systems as a primitive for implementing other distributed systems in th

A replicated log models the process of coming into consensus on the order of a series of values (generally numbe this, but the simplest and fastest is with a leader who chooses the ordering of values provided to it. As long as the ordering the leader chooses.

Of course if leaders didn't fail we wouldn't need followers! When the leader does die we need to choose a new leac behind or crash so we must ensure we choose an up-to-date follower. The fundamental guarantee a log replicatior committed, and the leader fails, the new leader we elect must also have that message. This yields a tradeoff: if the declaring it committed then there will be more potentially electable leaders.

If you choose the number of acknowledgements required and the number of logs that must be compared to elect called a Quorum.

A common approach to this tradeoff is to use a majority vote for both the commit decision and the leader election understand the tradeoffs. Let's say we have 2*f*+1 replicas. If *f*+1 replicas must receive a message prior to a comm electing the follower with the most complete log from at least *f*+1 replicas, then, with no more than *f* failures, the le because among any *f*+1 replicas, there must be at least one replica that contains all committed messages. That re as the new leader. There are many remaining details that each algorithm must handle (such as precisely defined w leader failure or changing the set of servers in the replica set) but we will ignore these for now.

This majority vote approach has a very nice property: the latency is dependent on only the fastest servers. That is, faster follower not the slower one.

There are a rich variety of algorithms in this family including ZooKeeper's [Zab](), [Raft](), and [Viewstamped Replication](). actual implementation is [PacificA]() from Microsoft.

The downside of majority vote is that it doesn't take many failures to leave you with no electable leaders. To tolera failures requires five copies of the data. In our experience having only enough redundancy to tolerate a single failu times, with 5x the disk space requirements and 1/5th the throughput, is not very practical for large volume data pr appear for shared cluster configuration such as ZooKeeper but are less common for primary data storage. For exa [majority-vote-based journal](), but this more expensive approach is not used for the data itself.

Kafka takes a slightly different approach to choosing its quorum set. Instead of majority vote, Kafka dynamically n leader. Only members of this set are eligible for election as leader. A write to a Kafka partition is not considered cc set is persisted to ZooKeeper whenever it changes. Because of this, any replica in the ISR is eligible to be elected l there are many partitions and ensuring leadership balance is important. With this ISR model and *f+1* replicas, a Ka messages.

For most use cases we hope to handle, we think this tradeoff is a reasonable one. In practice, to tolerate *f* failures, number of replicas to acknowledge before committing a message (e.g. to survive one failure a majority quorum ne approach requires two replicas and one acknowledgement). The ability to commit without the slowest servers is a ameliorated by allowing the client to choose whether they block on the message commit or not, and the additional factor is worth it.

Another important design distinction is that Kafka does not require that crashed nodes recover with all their data i depend on the existence of "stable storage" that cannot be lost in any failure-recovery scenario without potential c assumption. First, disk errors are the most common problem we observe in real operation of persistent data syste were not a problem, we do not want to require the use of fsync on every write for our consistency guarantees as th Our protocol for allowing a replica to rejoin the ISR ensures that before rejoining, it must fully re-sync again even if

Unclean leader election: What if they all die?

Note that Kafka's guarantee with respect to data loss is predicated on at least one replica remaining in sync. If all t

However a practical system needs to do something reasonable when all the replicas die. If you are unlucky enough There are two behaviors that could be implemented:

1. Wait for a replica in the ISR to come back to life and choose this replica as the leader (hopefully it still has all i
2. Choose the first replica (not necessarily in the ISR) that comes back to life as the leader.

This is a simple tradeoff between availability and consistency. If we wait for replicas in the ISR, then we will remain were destroyed or their data was lost, then we are permanently down. If, on the other hand, a non-in-sync replica co becomes the source of truth even though it is not guaranteed to have every committed message. By default from v for a consistent replica. This behavior can be changed using configuration property unclean.leader.election.enable

This dilemma is not specific to Kafka. It exists in any quorum-based scheme. For example in a majority voting sch must either choose to lose 100% of your data or violate consistency by taking what remains on an existing server a

Availability and Durability Guarantees

When writing to Kafka, producers can choose whether they wait for the message to be acknowledged by 0,1 or all guarantee that the full set of assigned replicas have received the message. By default, when acks=all, acknowledg received the message. For example, if a topic is configured with only two replicas and one fails (i.e., only one in syn However, these writes could be lost if the remaining replica also fails. Although this ensures maximum availability who prefer durability over availability. Therefore, we provide two topic-level configurations that can be used to pref

1. Disable unclean leader election - if all replicas become unavailable, then the partition will remain unavailable u effectively prefers unavailability over the risk of message loss. See the previous section on Unclean Leader El
2. Specify a minimum ISR size - the partition will only accept writes if the size of the ISR is above a certain minin just a single replica, which subsequently becomes unavailable. This setting only takes effect if the producer u acknowledged by at least this many in-sync replicas. This setting offers a trade-off between consistency and better consistency since the message is guaranteed to be written to more replicas which reduces the probabi partition will be unavailable for writes if the number of in-sync replicas drops below the minimum threshold.

Replica Management

The above discussion on replicated logs really covers only a single log, i.e. one topic partition. However a Kafka clu attempt to balance partitions within a cluster in a round-robin fashion to avoid clustering all partitions for high-volu leadership so that each node is the leader for a proportional share of its partitions.

It is also important to optimize the leadership election process as that is the critical window of unavailability. A nai election per partition for all partitions a node hosted when that node failed. Instead, we elect one of the brokers as and is responsible for changing the leader of all affected partitions in a failed broker. The result is that we are able notifications which makes the election process far cheaper and faster for a large number of partitions. If the contr controller.

4.8 Log Compaction

Log compaction ensures that Kafka will always retain at least the last known value for each message key within th and scenarios such as restoring state after application crashes or system failure, or reloading caches after applica use cases in more detail and then describe how compaction works.

So far we have described only the simpler approach to data retention where old log data is discarded after a fixed This works well for temporal event data such as logging where each record stands alone. However an important c (for example, the changes to a database table).

Let's discuss a concrete example of such a stream. Say we have a topic containing user email addresses; every tir topic using their user id as the primary key. Now say we send the following messages over some time period for a email address (messages for other ids are omitted):

```
123 => bill@microsoft.com
        .
        .
        .
123 => bill@gatesfoundation.org
        .
        .
        .
123 => bill@gmail.com
```

Log compaction gives us a more granular retention mechanism so that we are guaranteed to retain at least the las this we guarantee that the log contains a full snapshot of the final value for every key not just keys that changed re state off this topic without us having to retain a complete log of all changes.

Let's start by looking at a few use cases where this is useful, then we'll see how it can be used.

1. *Database change subscription*. It is often necessary to have a data set in multiple data systems, and often on perhaps a new-fangled key-value store). For example you might have a database, a cache, a search cluster, ar reflected in the cache, the search cluster, and eventually in Hadoop. In the case that one is only handling the r able to reload the cache or restore a failed search node you may need a complete data set.
2. *Event sourcing*. This is a style of application design which co-locates query processing with application desig application.
3. *Journaling for high-availability*. A process that does local computation can be made fault-tolerant by logging c reload these changes and carry on if it should fail. A concrete example of this is handling counts, aggregation Samza, a real-time stream-processing framework, uses this feature for exactly this purpose.

In each of these cases one needs primarily to handle the real-time feed of changes, but occasionally, when a mach
needs to do a full load. Log compaction allows feeding both of these use cases off the same backing topic. This s

The general idea is quite simple. If we had infinite log retention, and we logged each change in the above cases, th
from when it first began. Using this complete log, we could restore to any point in time by replaying the first N reco
for systems that update a single record many times as the log will grow without bound even for a stable dataset. T
updates will bound space but the log is no longer a way to restore the current state—now restoring from the begin
may not be captured at all.

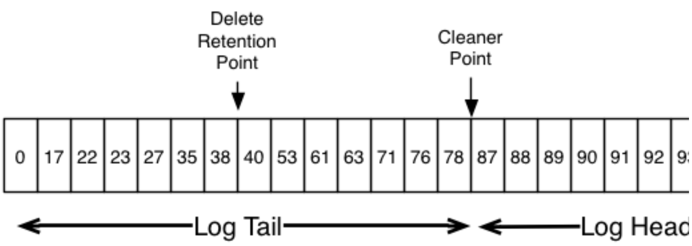
Log compaction is a mechanism to give finer-grained per-record retention, rather than the coarser-grained time-ba
have a more recent update with the same primary key. This way the log is guaranteed to have at least the last stat

This retention policy can be set per-topic, so a single cluster can have some topics where retention is enforced by
compaction.

This functionality is inspired by one of LinkedIn's oldest and most successful pieces of infrastructure—a database
structured storage systems Kafka is built for subscription and organizes data for fast linear reads and writes. Unlil
even in situations where the upstream data source would not otherwise be replayable.

Log Compaction Basics

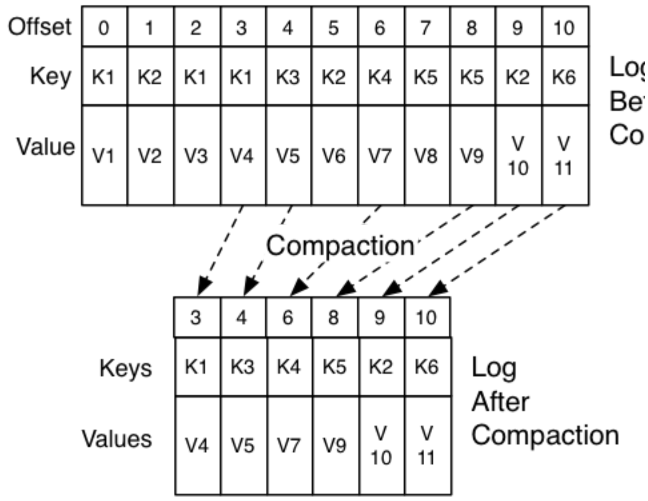
Here is a high-level picture that shows the logical structure of a Kafka log with the offset for each message.



The head of the log is identical to a traditional Kafka log. It has dense, sequential offsets and retains all messages
The picture above shows a log with a compacted tail. Note that the messages in the tail of the log retain the origin
changes. Note also that all offsets remain valid positions in the log, even if the message with that offset has been
from the next highest offset that does appear in the log. For example, in the picture above the offsets 36, 37, and 3
offsets would return a message set beginning with 38.

Compaction also allows for deletes. A message with a key and a null payload will be treated as a delete from the l
to be removed (as would any new message with that key), but delete markers are special in that they will themselv
The point in time at which deletes are no longer retained is marked as the "delete retention point" in the above diag

The compaction is done in the background by periodically recopying log segments. Cleaning does not block reads
I/O throughput to avoid impacting producers and consumers. The actual process of compacting a log segment lo



What guarantees does log compaction provide?

Log compaction guarantees the following:

1. Any consumer that stays caught-up to within the head of the log will see every message that is written; these `min.compaction.lag.ms` can be used to guarantee the minimum length of time must pass after a mess bound on how long each message will remain in the (uncompacted) head. The topic's `max.compaction.l` time a message is written and the time the message becomes eligible for compaction.
2. Ordering of messages is always maintained. Compaction will never re-order messages, just remove some.
3. The offset for a message never changes. It is the permanent identifier for a position in the log.
4. Any consumer progressing from the start of the log will see at least the final state of all records in the order tl will be seen, provided the consumer reaches the head of the log in a time period less than the topic's `delet` words: since the removal of delete markers happens concurrently with reads, it is possible for a consumer to `delete.retention.ms`.

Log Compaction Details

Log compaction is handled by the log cleaner, a pool of background threads that recopy log segment files, removir compactor thread works as follows:

1. It chooses the log that has the highest ratio of log head to log tail
2. It creates a succinct summary of the last offset for each key in the head of the log
3. It recopies the log from beginning to end removing keys which have a later occurrence in the log. New, clean s disk space required is just one additional log segment (not a fully copy of the log).
4. The summary of the log head is essentially just a space-compact hash table. It uses exactly 24 bytes per entr clean around 366GB of log head (assuming 1k messages).

Configuring The Log Cleaner

The log cleaner is enabled by default. This will start the pool of cleaner threads. To enable log cleaning on a partic

```
1  log.cleanup.policy=compact
```

The `log.cleanup.policy` property is a broker configuration setting defined in the broker's `server.prope`
have a configuration override in place as documented here. The log cleaner can be configured to retain a minimum
setting the compaction time lag.

```
1  log.cleaner.min.compaction.lag.ms
```

This can be used to prevent messages newer than a minimum message age from being subject to compaction. If
last segment, i.e. the one currently being written to. The active segment will not be compacted even if all of its me
cleaner can be configured to ensure a maximum delay after which the uncompacted "head" of the log becomes eli

```
1  log.cleaner.max.compaction.lag.ms
```

This can be used to prevent log with low produce rate from remaining ineligible for compaction for an unbounded
min.cleanable.dirty.ratio are not compacted. Note that this compaction deadline is not a hard guarantee since it is
actual compaction time. You will want to monitor the uncleanable-partitions-count, max-clean-time-secs and max-

Further cleaner configurations are described here.

4.9 Quotas

Kafka cluster has the ability to enforce quotas on requests to control the broker resources used by clients. Two typ
group of clients sharing a quota:

1. Network bandwidth quotas define byte-rate thresholds (since 0.9)
2. Request rate quotas define CPU utilization thresholds as a percentage of network and I/O threads (since 0.11

Why are quotas necessary?

It is possible for producers and consumers to produce/consume very high volumes of data or generate requests a
network saturation and generally DOS other clients and the brokers themselves. Having quotas protects against th
clusters where a small set of badly behaved clients can degrade user experience for the well behaved ones. In fact
enforce API limits according to an agreed upon contract.

Client groups

The identity of Kafka clients is the user principal which represents an authenticated user in a secure cluster. In a cl
grouping of unauthenticated users chosen by the broker using a configurable `PrincipalBuilder`. Client-id is
client application. The tuple (user, client-id) defines a secure logical group of clients that share both user principal

Quotas can be applied to (user, client-id), user or client-id groups. For a given connection, the most specific quota
group share the quota configured for the group. For example, if (user="test-user", client-id="test-client") has a prod
instances of user "test-user" with the client-id "test-client".

Quota Configuration

Quota configuration may be defined for (user, client-id), user and client-id groups. It is possible to override the defa lower) quota. The mechanism is similar to the per-topic log config overrides. User and (user, client-id) quota overri quota overrides are written under ***/config/clients***. These overrides are read by all brokers and are effective immedi restart of the entire cluster. See [here](#) for details. Default quotas for each group may also be updated dynamically u

The order of precedence for quota configuration is:

1. /config/users/<user>/clients/<client-id>
2. /config/users/<user>/clients/<default>
3. /config/users/<user>
4. /config/users/<default>/clients/<client-id>
5. /config/users/<default>/clients/<default>
6. /config/users/<default>
7. /config/clients/<client-id>
8. /config/clients/<default>

Broker properties (quota.producer.default, quota.consumer.default) can also be used to set defaults of network ba deprecated and will be removed in a later release. Default quotas for client-id can be set in Zookeeper similar to th

Network Bandwidth Quotas

Network bandwidth quotas are defined as the byte rate threshold for each group of clients sharing a quota. By def as configured by the cluster. This quota is defined on a per-broker basis. Each group of clients can publish/fetch a

Request Rate Quotas

Request rate quotas are defined as the percentage of time a client can utilize on request handler I/O threads and n `n%` represents `n%` of one thread, so the quota is out of a total capacity of `((num.io.threads + num.network.threads` upto `n%` across all I/O and network threads in a quota window before being throttled. Since the number of threads a number of cores available on the broker host, request rate quotas represent the total percentage of CPU that may l

Enforcement

By default, each unique client group receives a fixed quota as configured by the cluster. This quota is defined on a before it gets throttled. We decided that defining these quotas per broker is much better than having a fixed cluste mechanism to share client quota usage among all the brokers. This can be harder to get right than the quota imple

How does a broker react when it detects a quota violation? In our solution, the broker first computes the amount o returns a response with the delay immediately. In case of a fetch request, the response will not contain any data. T requests from the client anymore, until the delay is over. Upon receiving a response with a non-zero delay duration, the broker during the delay. Therefore, requests from a throttled client are effectively blocked from both sides. Eve response from the broker, the back pressure applied by the broker via muting its socket channel can still handle th further requests to the throttled channel will receive responses only after the delay is over.

Byte-rate and thread utilization are measured over multiple small windows (e.g. 30 windows of 1 second each) in (having large measurement windows (for e.g. 10 windows of 30 seconds each) leads to large bursts of traffic follo

5. IMPLEMENTATION

5.1 Network Layer

The network layer is a fairly straight-forward NIO server, and will not be described in great detail. The sendfile impl `writeTo` method. This allows the file-backed message set to use the more efficient `transferTo` implement is a single acceptor thread and *N* processor threads which handle a fixed number of connections each. This desig simple to implement and fast. The protocol is kept quite simple to allow for future implementation of clients in oth

5.2 Messages

Messages consist of a variable-length header, a variable length opaque key byte array and a variable length opaqu following section. Leaving the key and value opaque is the right decision: there is a great deal of progress being m unlikely to be right for all uses. Needless to say a particular application using Kafka would likely mandate a particu interface is simply an iterator over messages with specialized methods for bulk reading and writing to an NIO `Ch`

5.3 Message Format

Messages (aka Records) are always written in batches. The technical term for a batch of messages is a record ba degenerate case, we could have a record batch containing a single record. Record batches and records have their

5.3.1 Record Batch

The following is the on-disk format of a RecordBatch.

```
baseOffset: int64
batchLength: int32
partitionLeaderEpoch: int32
magic: int8 (current magic value is 2)
crc: int32
attributes: int16
    bit 0~2:
        0: no compression
        1: gzip
        2: snappy
        3: lz4
        4: zstd
    bit 3: timestampType
    bit 4: isTransactional (0 means not transactional)
    bit 5: isControlBatch (0 means not a control batch)
    bit 6~15: unused
lastOffsetDelta: int32
firstTimestamp: int64
maxTimestamp: int64
producerId: int64
producerEpoch: int16
```

```
baseSequence: int32
records: [Record]

```

Note that when compression is enabled, the compressed record data is serialized directly following the count of tl

The CRC covers the data from the attributes to the end of the batch (i.e. all the bytes that follow the CRC). It is loca magic byte before deciding how to interpret the bytes between the batch length and the magic byte. The partition l the need to recompute the CRC when this field is assigned for every batch that is received by the broker. The CRC-

On compaction: unlike the older message formats, magic v2 and above preserves the first and last offset/sequenc required in order to be able to restore the producer's state when the log is reloaded. If we did not retain the last sec producer might see an OutOfSequence error. The base sequence number must be preserved for duplicate checkin verifying that the first and last sequence numbers of the incoming batch match the last from that producer). As a r records in the batch are cleaned but batch is still retained in order to preserve a producer's last sequence number. during compaction, so it will change if the first record in the batch is compacted away.

5.3.1.1 Control Batches

A control batch contains a single record called the control record. Control records should not be passed on to appl transactional messages.

The key of a control record conforms to the following schema:

```
version: int16 (current version is 0)
type: int16 (0 indicates an abort marker, 1 indicates a commit)
```

The schema for the value of a control record is dependent on the type. The value is opaque to clients.

5.3.2 Record

Record level headers were introduced in Kafka 0.11.0. The on-disk format of a record with Headers is delineated b

```
length: varint
attributes: int8
    bit 0~7: unused
timestampDelta: varint
offsetDelta: varint
keyLength: varint
key: byte[]
valueLen: varint
value: byte[]
Headers => [Header]

```

5.3.2.1 Record Header

```
headerKeyLength: varint
headerKey: String
headerValueLength: varint
```

```
Value: byte[]

```

We use the same varint encoding as Protobuf. More information on the latter can be found here. The count of hea

5.3.3 Old Message Format

Prior to Kafka 0.11, messages were transferred and stored in *message sets*. In a message set, each message has represented as an array, they are not preceded by an int32 array size like other array elements in the protocol.

Message Set:

```
MessageSet (Version: 0) => [offset message_size message]
    offset => INT64
    message_size => INT32
    message => crc magic_byte attributes key value
        crc => INT32
        magic_byte => INT8
        attributes => INT8
            bit 0~2:
                0: no compression
                1: gzip
                2: snappy
            bit 3~7: unused
        key => BYTES
        value => BYTES
```

```
MessageSet (Version: 1) => [offset message_size message]
    offset => INT64
    message_size => INT32
    message => crc magic_byte attributes key value
        crc => INT32
        magic_byte => INT8
        attributes => INT8
            bit 0~2:
                0: no compression
                1: gzip
                2: snappy
                3: lz4
            bit 3: timestampType
                0: create time
                1: log append time
            bit 4~7: unused
        timestamp =>INT64
        key => BYTES
        value => BYTES
```

In versions prior to Kafka 0.10, the only supported message format version (which is indicated in the magic value) support in version 0.10.

- Similarly to version 2 above, the lowest bits of attributes represent the compression type.
- In version 1, the producer should always set the timestamp type bit to 0. If the topic is configured to use log ap log.message.timestamp.type = LogAppendTime or topic level config message.timestamp.type = LogAppendTi

timestamp in the message set.
- The highest bits of attributes must be set to 0.

In message format versions 0 and 1 Kafka supports recursive messages to enable compression. In this case the r compression types and the value field will contain a message set compressed with that type. We often refer to the as the "outer message." Note that the key should be null for the outer message and its offset will be the offset of tl

When receiving recursive version 0 messages, the broker decompresses them and each inner message is assigne compression, only the wrapper message will be assigned an offset. The inner messages will have relative offsets. outer message, which corresponds to the offset assigned to the last inner message.

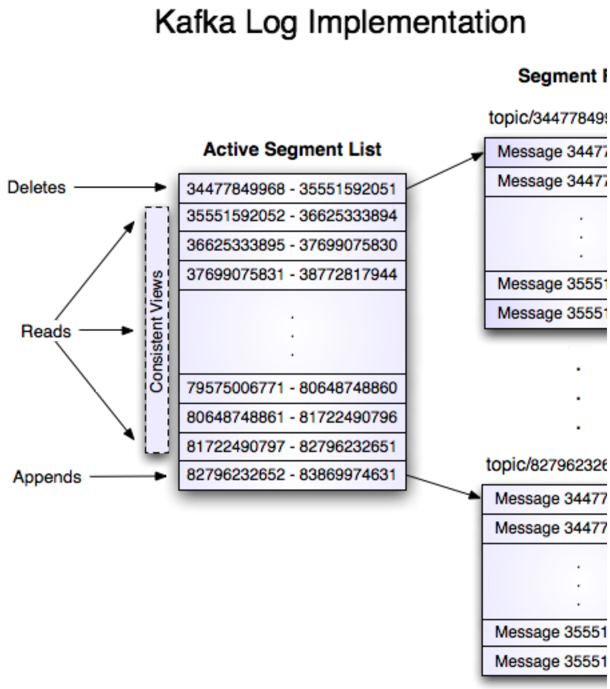
The crc field contains the CRC32 (and not CRC-32C) of the subsequent message bytes (i.e. from magic byte to the

5.4 Log

A log for a topic named "my_topic" with two partitions consists of two directories (namely `my_topic_0` and `m` for that topic. The format of the log files is a sequence of "log entries""; each log entry is a 4 byte integer *N* storing Each message is uniquely identified by a 64-bit integer *offset* giving the byte position of the start of this message i partition. The on-disk format of each message is given below. Each log file is named with the offset of the first me 00000000000.kafka, and each additional file will have an integer name roughly *S* bytes from the previous file wher

The exact binary format for records is versioned and maintained as a standard interface so record batches can be or conversion when desirable. The previous section included details about the on-disk format of records.

The use of the message offset as the message id is unusual. Our original idea was to use a GUID generated by the broker. But since a consumer must maintain an ID for each server, the global uniqueness of the GUID provides no v from a random id to an offset requires a heavy weight index structure which must be synchronized with disk, essen Thus to simplify the lookup structure we decided to use a simple per-partition atomic counter which could be coup message; this makes the lookup structure simpler, though multiple seeks per consumer request are still likely. How offset seemed natural—both after all are monotonically increasing integers unique to a partition. Since the offset is implementation detail and we went with the more efficient approach.



Writes

The log allows serial appends which always go to the last file. This file is rolled over to a fresh file when it reaches parameters: *M*, which gives the number of messages to write before forcing the OS to flush the file to disk, and *S*, gives a durability guarantee of losing at most *M* messages or *S* seconds of data in the event of a system crash.

Reads

Reads are done by giving the 64-bit logical offset of a message and an *S*-byte max chunk size. This will return an it intended to be larger than any single message, but in the event of an abnormally large message, the read can be re message is read successfully. A maximum message and buffer size can be specified to make the server reject me the maximum it needs to ever read to get a complete message. It is likely that the read buffer ends with a partial m

The actual process of reading from an offset requires first locating the log segment file in which the data is stored then reading from that file offset. The search is done as a simple binary search variation against an in-memory ran

The log provides the capability of getting the most recently written message to allow clients to start subscribing a to consume its data within its SLA-specified number of days. In this case when the client attempts to consume a r either reset itself or fail as appropriate to the use case.

The following is the format of the results sent to the consumer.

```
MessageSetSend (fetch result)

total length     : 4 bytes
error code       : 2 bytes
message 1        : x bytes
...
message n        : x bytes
```

```
MultiMessageSetSend (multiFetch result)

total length       : 4 bytes
error code         : 2 bytes
messageSetSend 1
...
messageSetSend n
```

Deletes

Data is deleted one log segment at a time. The log manager allows pluggable delete policies to choose which files modification time of more than *N* days ago, though a policy which retained the last *N* GB could also be useful. To a segment list we use a copy-on-write style segment list implementation that provides consistent views to allow a b the log segments while deletes are progressing.

Guarantees

The log provides a configuration parameter *M* which controls the maximum number of messages that are written run that iterates over all messages in the newest log segment and verifies that each message entry is valid. A mes length of the file AND the CRC32 of the message payload matches the CRC stored with the message. In the event

Note that two kinds of corruption must be handled: truncation in which an unwritten block is lost due to a crash, ar reason for this is that in general the OS makes no guarantee of the write order between the file inode and the actua nonsense data if the inode is updated with a new size but a crash occurs before the block containing that data is v corrupting the log (though the unwritten messages are, of course, lost).

5.5 Distribution

Consumer Offset Tracking

Kafka consumer tracks the maximum offset it has consumed in each partition and has the capability to commit of restart. Kafka provides the option to store all the offsets for a given consumer group in a designated broker (for th in that consumer group should send its offset commits and fetches to that group coordinator (broker). Consumer A consumer can look up its coordinator by issuing a FindCoordinatorRequest to any Kafka broker and reading the details. The consumer can then proceed to commit or fetch offsets from the coordinator broker. In case the coord Offset commits can be done automatically or manually by consumer instance.

When the group coordinator receives an OffsetCommitRequest, it appends the request to a special [compacted](#) Ka successful offset commit response to the consumer only after all the replicas of the offsets topic receive the offse timeout, the offset commit will fail and the consumer may retry the commit after backing off. The brokers periodic most recent offset commit per partition. The coordinator also caches the offsets in an in-memory table in order to

When the coordinator receives an offset fetch request, it simply returns the last committed offset vector from the became the coordinator for a new set of consumer groups (by becoming a leader for a partition of the offsets topi this case, the offset fetch will fail with an CoordinatorLoadInProgressException and the consumer may retry the O

ZooKeeper Directories

The following gives the ZooKeeper structures and algorithms used for co-ordination between consumers and brok

Notation

When an element in a path is denoted [xyz], that means that the value of xyz is not fixed and there is in fact a ZooK /topics/[topic] would be a directory named /topics containing a sub-directory for each topic name. Numerical rang 2, 3, 4. An arrow -> is used to indicate the contents of a znode. For example /hello -> world would indicate a znode

Broker Node Registry

```
/brokers/ids/[0...N] --> {"jmx_port":...,"timestamp":...,"endpoints":[...],"host":...
```

This is a list of all present broker nodes, each of which provides a unique logical broker id which identifies it to con startup, a broker node registers itself by creating a znode with the logical broker id under /brokers/ids. The purpos different physical machine without affecting consumers. An attempt to register a broker id that is already in use (s results in an error.

Since the broker registers itself in ZooKeeper using ephemeral znodes, this registration is dynamic and will disapp no longer available).

Broker Topic Registry

```
/brokers/topics/[topic]/partitions/[0...N]/state --> {"controller_epoch":...,"leader"
```

Each broker registers itself under the topics it maintains and stores the number of partitions for that topic.

Cluster Id

The cluster id is a unique and immutable identifier assigned to a Kafka cluster. The cluster id can have a maximum regular expression [a-zA-Z0-9_\-]+, which corresponds to the characters used by the URL-safe Base64 variant with started for the first time.

Implementation-wise, it is generated when a broker with version 0.10.1 or later is successfully started for the first t `/cluster/id` znode during startup. If the znode does not exist, the broker generates a new cluster id and crea

Broker node registration

The broker nodes are basically independent, so they only publish information about what they have. When a broker and writes information about its host name and port. The broker also register the list of existing topics and their lo registered dynamically when they are created on the broker.

6. OPERATIONS

Here is some information on actually running Kafka as a production system based on usage and experience at Lin

6.1 Basic Kafka Operations

This section will review the most common operations you will perform on your Kafka cluster. All of the tools reviev Kafka distribution and each tool will print details on all possible commandline options if it is run with no argument

Adding and removing topics

You have the option of either adding topics manually or having them be created automatically when data is first pu may want to tune the default topic configurations used for auto-created topics.

Topics are added and modified using the topic tool:

```
> bin/kafka-topics.sh --bootstrap-server broker_host:port --create --topic my_topic_n
      --partitions 20 --replication-factor 3 --config x=y
```

The replication factor controls how many servers will replicate each message that is written. If you have a replicati access to your data. We recommend you use a replication factor of 2 or 3 so that you can transparently bounce m

The partition count controls how many logs the topic will be sharded into. There are several impacts of the partitic if you have 20 partitions the full data set (and read and write load) will be handled by no more than 20 servers (not maximum parallelism of your consumers. This is discussed in greater detail in the concepts section.

Each sharded partition log is placed into its own folder under the Kafka log directory. The name of such folders co id. Since a typical folder name can not be over 255 characters long, there will be a limitation on the length of topic 100,000. Therefore, topic names cannot be longer than 249 characters. This leaves just enough room in the folder

The configurations added on the command line override the default settings the server has for things like the lengt configurations is documented here.

Modifying topics

You can change the configuration or partitioning of a topic using the same topic tool.

To add partitions you can do

```
> bin/kafka-topics.sh --bootstrap-server broker_host:port --alter --topic my_topic_na
      --partitions 40
```

Be aware that one use case for partitions is to semantically partition data, and adding partitions doesn't change th they rely on that partition. That is if data is partitioned by `hash(key) % number_of_partitions` then this p Kafka will not attempt to automatically redistribute data in any way.

To add configs:

```
> bin/kafka-configs.sh --bootstrap-server broker_host:port --entity-type topics --ent
```

To remove a config:

```
> bin/kafka-configs.sh --bootstrap-server broker_host:port --entity-type topics --ent
```

And finally deleting a topic:

```
> bin/kafka-topics.sh --bootstrap-server broker_host:port --delete --topic my_topic_n
```

Kafka does not currently support reducing the number of partitions for a topic.

Instructions for changing the replication factor of a topic can be found here.

Graceful shutdown

The Kafka cluster will automatically detect any broker shutdown or failure and elect new leaders for the partitions
brought down intentionally for maintenance or configuration changes. For the latter cases Kafka supports a more
a server is stopped gracefully it has two optimizations it will take advantage of:

1. It will sync all its logs to disk to avoid needing to do any log recovery when it restarts (i.e. validating the check
time so this speeds up intentional restarts.
2. It will migrate any partitions the server is the leader for to other replicas prior to shutting down. This will make
is unavailable to a few milliseconds.

Syncing the logs will happen automatically whenever the server is stopped other than by a hard kill, but the control

```
controlled.shutdown.enable=true
```

Note that controlled shutdown will only succeed if *all* the partitions hosted on the broker have replicas (i.e. the rep
alive). This is generally what you want since shutting down the last replica would make that topic partition unavail

Balancing leadership

Whenever a broker stops or crashes leadership for that broker's partitions transfers to other replicas. This means t
for all its partitions, meaning it will not be used for client reads and writes.

To avoid this imbalance, Kafka has a notion of preferred replicas. If the list of replicas for a partition is 1,5,9 then n
earlier in the replica list. You can have the Kafka cluster try to restore leadership to the restored replicas by running

```
> bin/kafka-preferred-replica-election.sh --zookeeper zk_host:port/chroot
```

Since running this command can be tedious you can also configure Kafka to do this automatically by setting the fo

```
auto.leader.rebalance.enable=true
```

Balancing Replicas Across Racks

The rack awareness feature spreads replicas of the same partition across different racks. This extends the guaran
the risk of data loss should all the brokers on a rack fail at once. The feature can also be applied to other broker gr

You can specify that a broker belongs to a particular rack by adding a property to the broker config:

```
broker.rack=my-rack-id
```

When a topic is created, modified or replicas are redistributed, the rack constraint will be honoured, ensuring replic min(#racks, replication-factor) different racks).

The algorithm used to assign replicas to brokers ensures that the number of leaders per broker will be constant, re balanced throughput.

However if racks are assigned different numbers of brokers, the assignment of replicas will not be even. Racks wit storage and put more resources into replication. Hence it is sensible to configure an equal number of brokers per r

Mirroring data between clusters

We refer to the process of replicating data *between* Kafka clusters "mirroring" to avoid confusion with the replicati comes with a tool for mirroring data between Kafka clusters. The tool consumes from a source cluster and produc mirroring is to provide a replica in another datacenter. This scenario will be discussed in more detail in the next se

You can run many such mirroring processes to increase throughput and for fault-tolerance (if one process dies, th

Data will be read from topics in the source cluster and written to a topic with the same name in the destination clu and producer hooked together.

The source and destination clusters are completely independent entities: they can have different numbers of parti mirror cluster is not really intended as a fault-tolerance mechanism (as the consumer position will be different); fo mirror maker process will, however, retain and use the message key for partitioning so order is preserved on a per-

Here is an example showing how to mirror a single topic (named *my-topic*) from an input cluster:

```
> bin/kafka-mirror-maker.sh
      --consumer.config consumer.properties
      --producer.config producer.properties --whitelist my-topic
```

Note that we specify the list of topics with the `--whitelist` option. This option allows any regular expression topics named *A* and *B* using `--whitelist 'A|B'` . Or you could mirror *all* topics using `--whitelist '*'` doesn't try to expand it as a file path. For convenience we allow the use of ',' instead of '|' to specify a list of topics. `auto.create.topics.enable=true` makes it possible to have a replica cluster that will automatically creat added.

Checking consumer position

Sometimes it's useful to see the position of your consumers. We have a tool that will show the position of all cons the log they are. To run this tool on a consumer group named *my-group* consuming a topic named *my-topic* would

```
> bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group m

TOPIC                          PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG
my-topic                       0          2               4               2
my-topic                       1          2               3               1
my-topic                       2          2               3               1
```

Managing Consumer Groups

With the ConsumerGroupCommand tool, we can list, describe, or delete the consumer groups. The consumer grou
committed offset for that group expires. Manual deletion works only if the group does not have any active membe

```
> bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --list

test-consumer-group
```

To view offsets, as mentioned earlier, we "describe" the consumer group like this:

```
> bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group m

TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-I
topic3          0          241019          395308          154289          consumer2-
topic2          1          520678          803288          282610          consumer2-
topic3          1          241018          398817          157799          consumer2-
topic1          0          854144          855809          1665            consumer1-
topic2          0          460537          803290          342753          consumer1-
topic3          2          243655          398812          155157          consumer4-
```

There are a number of additional "describe" options that can be used to provide more detailed information about a

- –members: This option provides the list of all active members in the consumer group.

  ```
  > bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --grour

  CONSUMER-ID                                    HOST            CLIENT-ID       #PAF
  consumer1-3fc8d6f1-581a-4472-bdf3-3515b4aee8c1 /127.0.0.1      consumer1       2
  consumer4-117fe4d3-c6c1-4178-8ee9-eb4a3954bee0 /127.0.0.1      consumer4       1
  consumer2-e76ea8c3-5d30-4299-9005-47eb41f3d3c4 /127.0.0.1      consumer2       3
  consumer3-ecea43e4-1f01-479f-8349-f9130b75d8ee /127.0.0.1      consumer3       0
  ```

- –members –verbose: On top of the information reported by the "–members" options above, this option also pro

  ```
  > bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --grour

  CONSUMER-ID                                    HOST            CLIENT-ID       #PAF
  consumer1-3fc8d6f1-581a-4472-bdf3-3515b4aee8c1 /127.0.0.1      consumer1       2
  consumer4-117fe4d3-c6c1-4178-8ee9-eb4a3954bee0 /127.0.0.1      consumer4       1
  consumer2-e76ea8c3-5d30-4299-9005-47eb41f3d3c4 /127.0.0.1      consumer2       3
  consumer3-ecea43e4-1f01-479f-8349-f9130b75d8ee /127.0.0.1      consumer3       0
  ```

- –offsets: This is the default describe option and provides the same output as the "–describe" option.
- –state: This option provides useful group-level information.

  ```
  > bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --grour

  COORDINATOR (ID)         ASSIGNMENT-STRATEGY       STATE                #MEMBERS
  localhost:9092 (0)       range                     Stable               4
  ```

To manually delete one or multiple consumer groups, the "–delete" option can be used:

```
> bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --delete --group my-

Deletion of requested consumer groups ('my-group', 'my-other-group') was successful.
```

To reset offsets of a consumer group, "--reset-offsets" option can be used. This option supports one consumer grc
-topic. One scope must be selected, unless you use '--from-file' scenario. Also, first make sure that the consumer ir

It has 3 execution options:

- (default) to display which offsets to reset.
- --execute : to execute --reset-offsets process.
- --export : to export the results to a CSV format.

--reset-offsets also has following scenarios to choose from (atleast one scenario must be selected):

- --to-datetime <String: datetime> : Reset offsets to offsets from datetime. Format: 'YYYY-MM-DDTHH:mm:SS.ss
- --to-earliest : Reset offsets to earliest offset.
- --to-latest : Reset offsets to latest offset.
- --shift-by <Long: number-of-offsets> : Reset offsets shifting current offset by 'n', where 'n' can be positive or neg
- --from-file : Reset offsets to values defined in CSV file.
- --to-current : Resets offsets to current offset.
- --by-duration <String: duration> : Reset offsets to offset by duration from current timestamp. Format: 'PnDTnHn
- --to-offset : Reset offsets to a specific offset.

Please note, that out of range offsets will be adjusted to available offset end. For example, if offset end is at 10 an
selected.

For example, to reset offsets of a consumer group to the latest offset:

```
> bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --reset-offsets --gr

TOPIC                          PARTITION  NEW-OFFSET
topic1                         0          0
```

If you are using the old high-level consumer and storing the group metadata in ZooKeeper (i.e. `offsets.storag`
`bootstrap-server` :

```
> bin/kafka-consumer-groups.sh --zookeeper localhost:2181 --list
```

Expanding your cluster

Adding servers to a Kafka cluster is easy, just assign them a unique broker id and start up Kafka on your new serve
any data partitions, so unless partitions are moved to them they won't be doing any work until new topics are creat
want to migrate some existing data to these machines.

The process of migrating data is manually initiated but fully automated. Under the covers what happens is that Ka
migrating and allow it to fully replicate the existing data in that partition. When the new server has fully replicated t
the existing replicas will delete their partition's data.

The partition reassignment tool can be used to move partitions across brokers. An ideal partition distribution wou
partition reassignment tool does not have the capability to automatically study the data distribution in a Kafka clus
As such, the admin has to figure out which topics or partitions should be moved around.

The partition reassignment tool can run in 3 mutually exclusive modes:

- --generate: In this mode, given a list of topics and a list of brokers, the tool generates a candidate reassignment This option merely provides a convenient way to generate a partition reassignment plan given a list of topics ar
- --execute: In this mode, the tool kicks off the reassignment of partitions based on the user provided reassignme be a custom reassignment plan hand crafted by the admin or provided by using the --generate option
- --verify: In this mode, the tool verifies the status of the reassignment for all partitions listed during the last --exe in progress

Automatically migrating data to new machines

The partition reassignment tool can be used to move some topics off of the current set of brokers to the newly ad cluster since it is easier to move entire topics to the new set of brokers, than moving one partition at a time. When be moved to the new set of brokers and a target list of new brokers. The tool then evenly distributes all partitions f move, the replication factor of the topic is kept constant. Effectively the replicas for all partitions for the input list c brokers.

For instance, the following example will move all partitions for topics foo1,foo2 to the new set of brokers 5,6. At th exist on brokers 5,6.

Since the tool accepts the input list of topics as a json file, you first need to identify the topics you want to move a

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

Once the json file is ready, use the partition reassignment tool to generate a candidate assignment:

```
> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --topics-to-move-json-
Current partition replica assignment

{"version":1,
"partitions":[{"topic":"foo1","partition":2,"replicas":[1,2]},
              {"topic":"foo1","partition":0,"replicas":[3,4]},
              {"topic":"foo2","partition":2,"replicas":[1,2]},
              {"topic":"foo2","partition":0,"replicas":[3,4]},
              {"topic":"foo1","partition":1,"replicas":[2,3]},
              {"topic":"foo2","partition":1,"replicas":[2,3]}]
}

Proposed partition reassignment configuration

{"version":1,
"partitions":[{"topic":"foo1","partition":2,"replicas":[5,6]},
              {"topic":"foo1","partition":0,"replicas":[5,6]},
              {"topic":"foo2","partition":2,"replicas":[5,6]},
              {"topic":"foo2","partition":0,"replicas":[5,6]},
              {"topic":"foo1","partition":1,"replicas":[5,6]},
              {"topic":"foo2","partition":1,"replicas":[5,6]}]
```

```
}
```

The tool generates a candidate assignment that will move all partitions from topics foo1,foo2 to brokers 5,6. Note it merely tells you the current assignment and the proposed new assignment. The current assignment should be s be saved in a json file (e.g. expand-cluster-reassignment.json) to be input to the tool with the --execute option as f

```
> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-fi
Current partition replica assignment

{"version":1,
"partitions":[{"topic":"foo1","partition":2,"replicas":[1,2]},
              {"topic":"foo1","partition":0,"replicas":[3,4]},
              {"topic":"foo2","partition":2,"replicas":[1,2]},
              {"topic":"foo2","partition":0,"replicas":[3,4]},
              {"topic":"foo1","partition":1,"replicas":[2,3]},
              {"topic":"foo2","partition":1,"replicas":[2,3]}]
}

Save this to use as the --reassignment-json-file option during rollback
Successfully started reassignment of partitions
{"version":1,
"partitions":[{"topic":"foo1","partition":2,"replicas":[5,6]},
              {"topic":"foo1","partition":0,"replicas":[5,6]},
              {"topic":"foo2","partition":2,"replicas":[5,6]},
              {"topic":"foo2","partition":0,"replicas":[5,6]},
              {"topic":"foo1","partition":1,"replicas":[5,6]},
              {"topic":"foo2","partition":1,"replicas":[5,6]}]
}
```

Finally, the --verify option can be used with the tool to check the status of the partition reassignment. Note that the option) should be used with the --verify option:

```
> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-fil
Status of partition reassignment:
Reassignment of partition [foo1,0] completed successfully
Reassignment of partition [foo1,1] is in progress
Reassignment of partition [foo1,2] is in progress
Reassignment of partition [foo2,0] completed successfully
Reassignment of partition [foo2,1] completed successfully
Reassignment of partition [foo2,2] completed successfully
```

Custom partition assignment and migration

The partition reassignment tool can also be used to selectively move replicas of a partition to a specific set of bro the reassignment plan and does not require the tool to generate a candidate reassignment, effectively skipping the

For instance, the following example moves partition 0 of topic foo1 to brokers 5,6 and partition 1 of topic foo2 to b

The first step is to hand craft the custom reassignment plan in a json file:

```
> cat custom-reassignment.json
{"version":1,"partitions":[{"topic":"foo1","partition":0,"replicas":[5,6]},{"topic":"
```

Then, use the json file with the –execute option to start the reassignment process:

```
> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-fi
Current partition replica assignment

{"version":1,
"partitions":[{"topic":"foo1","partition":0,"replicas":[1,2]},
              {"topic":"foo2","partition":1,"replicas":[3,4]}]
}

Save this to use as the --reassignment-json-file option during rollback
Successfully started reassignment of partitions
{"version":1,
"partitions":[{"topic":"foo1","partition":0,"replicas":[5,6]},
              {"topic":"foo2","partition":1,"replicas":[2,3]}]
}
```

The –verify option can be used with the tool to check the status of the partition reassignment. Note that the same be used with the –verify option:

```
> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-fil
Status of partition reassignment:
Reassignment of partition [foo1,0] completed successfully
Reassignment of partition [foo2,1] completed successfully
```

Decommissioning brokers

The partition reassignment tool does not have the ability to automatically generate a reassignment plan for decom reassignment plan to move the replica for all partitions hosted on the broker to be decommissioned, to the rest of needs to ensure that all the replicas are not moved from the decommissioned broker to only one other broker. To r decommissioning brokers in the future.

Increasing replication factor

Increasing the replication factor of an existing partition is easy. Just specify the extra replicas in the custom reass the replication factor of the specified partitions.

For instance, the following example increases the replication factor of partition 0 of topic foo from 1 to 3. Before in on broker 5. As part of increasing the replication factor, we will add more replicas on brokers 6 and 7.

The first step is to hand craft the custom reassignment plan in a json file:

```
> cat increase-replication-factor.json
{"version":1,
"partitions":[{"topic":"foo","partition":0,"replicas":[5,6,7]}]}
```

Then, use the json file with the –execute option to start the reassignment process:

```
> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-fi
Current partition replica assignment

{"version":1,
```

```
"partitions":[{"topic":"foo","partition":0,"replicas":[5]}]}

Save this to use as the --reassignment-json-file option during rollback
Successfully started reassignment of partitions
{"version":1,
"partitions":[{"topic":"foo","partition":0,"replicas":[5,6,7]}]}
```

The –verify option can be used with the tool to check the status of the partition reassignment. Note that the same should be used with the –verify option:

```
> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-fil
Status of partition reassignment:
Reassignment of partition [foo,0] completed successfully
```

You can also verify the increase in replication factor with the kafka-topics tool:

```
> bin/kafka-topics.sh --bootstrap-server localhost:9092 --topic foo --describe
Topic:foo   PartitionCount:1    ReplicationFactor:3 Configs:
  Topic: foo    Partition: 0    Leader: 5   Replicas: 5,6,7 Isr: 5,6,7
```

Limiting Bandwidth Usage during Data Migration

Kafka lets you apply a throttle to replication traffic, setting an upper bound on the bandwidth used to move replica cluster, bootstrapping a new broker or adding or removing brokers, as it limits the impact these data-intensive ope

There are two interfaces that can be used to engage a throttle. The simplest, and safest, is to apply a throttle wher can also be used to view and alter the throttle values directly.

So for example, if you were to execute a rebalance, with the below command, it would move partitions at no more

```
$ bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --execute --reassignmen
```

When you execute this script you will see the throttle engage:

```
The throttle limit was set to 50000000 B/s
Successfully started reassignment of partitions.
```

Should you wish to alter the throttle, during a rebalance, say to increase the throughput so it completes quicker, yo same reassignment-json-file:

```
$ bin/kafka-reassign-partitions.sh --zookeeper localhost:2181  --execute --reassignme
  There is an existing assignment running.
  The throttle limit was set to 700000000 B/s
```

Once the rebalance completes the administrator can check the status of the rebalance using the –verify option. If verify command. It is important that administrators remove the throttle in a timely manner once rebalancing comp so could cause regular replication traffic to be throttled.

When the –verify option is executed, and the reassignment has completed, the script will confirm that the throttle

```
> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181  --verify --reassignmen
Status of partition reassignment:
Reassignment of partition [my-topic,1] completed successfully
Reassignment of partition [mytopic,0] completed successfully
```

```
Throttle was removed.
```

The administrator can also validate the assigned configs using the kafka-configs.sh. There are two pairs of throttl value itself. This is configured, at a broker level, using the dynamic properties:

```
leader.replication.throttled.rate
  follower.replication.throttled.rate
```

There is also an enumerated set of throttled replicas:

```
leader.replication.throttled.replicas
  follower.replication.throttled.replicas
```

Which are configured per topic. All four config values are automatically assigned by kafka-reassign-partitions.sh (

To view the throttle limit configuration:

```
> bin/kafka-configs.sh --describe --zookeeper localhost:2181 --entity-type brokers
Configs for brokers '2' are leader.replication.throttled.rate=700000000,follower.repl
Configs for brokers '1' are leader.replication.throttled.rate=700000000,follower.repl
```

This shows the throttle applied to both leader and follower side of the replication protocol. By default both sides a

To view the list of throttled replicas:

```
> bin/kafka-configs.sh --describe --zookeeper localhost:2181 --entity-type topics
Configs for topic 'my-topic' are leader.replication.throttled.replicas=1:102,0:101,
    follower.replication.throttled.replicas=1:101,0:102
```

Here we see the leader throttle is applied to partition 1 on broker 102 and partition 0 on broker 101. Likewise the f 0 on broker 102.

By default kafka-reassign-partitions.sh will apply the leader throttle to all replicas that exist before the rebalance, a all move destinations. So if there is a partition with replicas on brokers 101,102, being reassigned to 102,103, a lea follower throttle would be applied to 103 only.

If required, you can also use the --alter switch on kafka-configs.sh to alter the throttle configurations manually.

Safe usage of throttled replication

Some care should be taken when using throttled replication. In particular:

(1) Throttle Removal:

The throttle should be removed in a timely manner once reassignment completes (by running kafka-reassign-parti

(2) Ensuring Progress:

If the throttle is set too low, in comparison to the incoming write rate, it is possible for replication to not make prog

```
max(BytesInPerSec) > throttle
```

Where BytesInPerSec is the metric that monitors the write throughput of producers into each broker.

The administrator can monitor whether replication is making progress, during the rebalance, using the metric:

```
kafka.server:type=FetcherLagMetrics,name=ConsumerLag,clientId=([-.\w]+),topic=([-
```

The lag should constantly decrease during replication. If the metric does not decrease the administrator should in

Setting quotas

Quotas overrides and defaults may be configured at (user, client-id), user or client-id levels as described here. By d custom quotas for each (user, client-id), user or client-id group.

Configure custom quota for (user=user1, client-id=clientA):

```
> bin/kafka-configs.sh  --zookeeper localhost:2181 --alter --add-config 'producer_byt
Updated config for entity: user-principal 'user1', client-id 'clientA'.
```

Configure custom quota for user=user1:

```
> bin/kafka-configs.sh  --zookeeper localhost:2181 --alter --add-config 'producer_byt
Updated config for entity: user-principal 'user1'.
```

Configure custom quota for client-id=clientA:

```
> bin/kafka-configs.sh  --zookeeper localhost:2181 --alter --add-config 'producer_byt
Updated config for entity: client-id 'clientA'.
```

It is possible to set default quotas for each (user, client-id), user or client-id group by specifying *--entity-default* opt

Configure default client-id quota for user=userA:

```
> bin/kafka-configs.sh  --zookeeper localhost:2181 --alter --add-config 'producer_byt
Updated config for entity: user-principal 'user1', default client-id.
```

Configure default quota for user:

```
> bin/kafka-configs.sh  --zookeeper localhost:2181 --alter --add-config 'producer_byt
Updated config for entity: default user-principal.
```

Configure default quota for client-id:

```
> bin/kafka-configs.sh  --zookeeper localhost:2181 --alter --add-config 'producer_byt
Updated config for entity: default client-id.
```

Here's how to describe the quota for a given (user, client-id):

```
> bin/kafka-configs.sh  --zookeeper localhost:2181 --describe --entity-type users --e
Configs for user-principal 'user1', client-id 'clientA' are producer_byte_rate=1024,c
```

Describe quota for a given user:

```
> bin/kafka-configs.sh  --zookeeper localhost:2181 --describe --entity-type users --e
Configs for user-principal 'user1' are producer_byte_rate=1024,consumer_byte_rate=204
```

Describe quota for a given client-id:

```
> bin/kafka-configs.sh  --zookeeper localhost:2181 --describe --entity-type clients -
Configs for client-id 'clientA' are producer_byte_rate=1024,consumer_byte_rate=2048,r
```

If entity name is not specified, all entities of the specified type are described. For example, describe all users:

```
> bin/kafka-configs.sh  --zookeeper localhost:2181 --describe --entity-type users
Configs for user-principal 'user1' are producer_byte_rate=1024,consumer_byte_rate=204
Configs for default user-principal are producer_byte_rate=1024,consumer_byte_rate=204
```

Similarly for (user, client):

```
> bin/kafka-configs.sh  --zookeeper localhost:2181 --describe --entity-type users --e
Configs for user-principal 'user1', default client-id are producer_byte_rate=1024,con
Configs for user-principal 'user1', client-id 'clientA' are producer_byte_rate=1024,c
```

It is possible to set default quotas that apply to all client-ids by setting these configs on the brokers. These propert
configured in Zookeeper. By default, each client-id receives an unlimited quota. The following sets the default quot

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

Note that these properties are being deprecated and may be removed in a future release. Defaults configured usin

6.2 Datacenters

Some deployments will need to manage a data pipeline that spans multiple datacenters. Our recommended appro
with application instances in each datacenter interacting only with their local cluster and mirroring between cluste
this).

This deployment pattern allows datacenters to act as independent entities and allows us to manage and tune inte
alone and operate even if the inter-datacenter links are unavailable: when this occurs the mirroring falls behind unt

For applications that need a global view of all data you can use mirroring to provide clusters which have aggregate
aggregate clusters are used for reads by applications that require the full data set.

This is not the only possible deployment pattern. It is possible to read from or write to a remote Kafka cluster over
required to get the cluster.

Kafka naturally batches data in both the producer and consumer so it can achieve high-throughput even over a hig
increase the TCP socket buffer sizes for the producer, consumer, and broker using the `socket.send.buffer.l`
The appropriate way to set this is documented [here](#).

It is generally *not* advisable to run a *single* Kafka cluster that spans multiple datacenters over a high-latency link. T
ZooKeeper writes, and neither Kafka nor ZooKeeper will remain available in all locations if the network between lo

6.3 Kafka Configuration

Important Client Configurations

The most important producer configurations are:

- acks

- compression
- batch size

The most important consumer configuration is the fetch size.

All configurations are documented in the configuration section.

A Production Server Config

Here is an example production server configuration:

```
# ZooKeeper
zookeeper.connect=[list of ZooKeeper servers]

# Log configuration
num.partitions=8
default.replication.factor=3
log.dir=[List of directories. Kafka should have its own dedicated disk(s) or SSD(s).

# Other configurations
broker.id=[An integer. Start with 0 and increment by 1 for each new broker.]
listeners=[list of listeners]
auto.create.topics.enable=false
min.insync.replicas=2
queued.max.requests=[number of concurrent requests]
```

Our client configuration varies a fair amount between different use cases.

6.4 Java Version

From a security perspective, we recommend you use the latest released version of JDK 1.8 as older freely availabl
currently running JDK 1.8 u5 (looking to upgrade to a newer version) with the G1 collector. LinkedIn's tuning looks

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

For reference, here are the stats on one of LinkedIn's busiest clusters (at peak):

- 60 brokers
- 50k partitions (replication factor 2)
- 800k messages/sec in
- 300 MB/sec inbound, 1 GB/sec+ outbound

The tuning looks fairly aggressive, but all of the brokers in that cluster have a 90% GC pause time of about 21ms, a

6.5 Hardware and OS

We are using dual quad-core Intel Xeon machines with 24GB of memory.

You need sufficient memory to buffer active readers and writers. You can do a back-of-the-envelope estimate of m
seconds and compute your memory need as write_throughput*30.

The disk throughput is important. We have 8x7200 rpm SATA drives. In general disk throughput is the performance configure flush behavior you may or may not benefit from more expensive disks (if you force flush often then highe

OS

Kafka should run well on any unix system and has been tested on Linux and Solaris.

We have seen a few issues running on Windows and Windows is not currently a well supported platform though w

It is unlikely to require much OS-level tuning, but there are three potentially important OS-level configurations:

- File descriptor limits: Kafka uses file descriptors for log segments and open connections. If a broker hosts man (number_of_partitions)*(partition_size/segment_size) to track all log segments in addition to the number of co allowed file descriptors for the broker processes as a starting point. Note: The mmap() function adds an extra r is not removed by a subsequent close() on that file descriptor. This reference is removed when there are no mo
- Max socket buffer size: can be increased to enable high-performance data transfer between data centers as de
- Maximum number of memory map areas a process may have (aka vm.max_map_count). See the Linux kernel c when considering the maximum number of partitions a broker may have. By default, on a number of Linux syste Each log segment, allocated per partition, requires a pair of index/timeindex files, and each of these files consu areas. Thus, each partition requires minimum 2 map areas, as long as it hosts a single log segment. That is to s 100000 map areas and likely cause broker crash with OutOfMemoryError (Map failed) on a system with default segments per partition varies depending on the segment size, load intensity, retention policy and, generally, ten

Disks and Filesystem

We recommend using multiple drives to get good throughput and not sharing the same drives used for Kafka data good latency. You can either RAID these drives together into a single volume or format and mount each drive as its provided by RAID can also be provided at the application level. This choice has several tradeoffs.

If you configure multiple data directories partitions will be assigned round-robin to data directories. Each partition balanced among partitions this can lead to load imbalance between disks.

RAID can potentially do better at balancing load between disks (although it doesn't always seem to) because it bal is usually a big performance hit for write throughput and reduces the available disk space.

Another potential benefit of RAID is the ability to tolerate disk failures. However our experience has been that rebu the server, so this does not provide much real availability improvement.

Application vs. OS Flush Management

Kafka always immediately writes all data to the filesystem and supports the ability to configure the flush policy tha using the flush. This flush policy can be controlled to force data to disk after a period of time or after a certain nun this configuration.

Kafka must eventually call fsync to know that data was flushed. When recovering from a crash for any log segmen message by checking its CRC and also rebuild the accompanying offset index file as part of the recovery process

Note that durability in Kafka does not require syncing data to disk, as a failed node will always recover from its rep

We recommend using the default flush settings which disable application fsync entirely. This means relying on the flush. This provides the best of all worlds for most uses: no knobs to tune, great throughput and latency, and full re by replication are stronger than sync to local disk, however the paranoid still may prefer having both and applicatic

The drawback of using application level flush settings is that it is less efficient in its disk usage pattern (it gives th fsync in most Linux filesystems blocks writes to the file whereas the background flushing does much more granul

In general you don't need to do any low-level tuning of the filesystem, but in the next few sections we will go over s

Understanding Linux OS Flush Behavior

In Linux, data written to the filesystem is maintained in [pagecache]() until it must be written out to disk (due to an ap data is done by a set of background threads called pdflush (or in post 2.6.32 kernels "flusher threads").

Pdflush has a configurable policy that controls how much dirty data can be maintained in cache and for how long When Pdflush cannot keep up with the rate of data being written it will eventually cause the writing process to bloc data.

You can see the current state of OS memory usage by doing

```
> cat /proc/meminfo
```

The meaning of these values are described in the link above.

Using pagecache has several advantages over an in-process cache for storing data that will be written out to disk:

- The I/O scheduler will batch together consecutive small writes into bigger physical writes which improves throu
- The I/O scheduler will attempt to re-sequence writes to minimize movement of the disk head which improves th
- It automatically uses all the free memory on the machine

Filesystem Selection

Kafka uses regular files on disk, and as such it has no hard dependency on a specific filesystem. The two filesyste Historically, EXT4 has had more usage, but recent improvements to the XFS filesystem have shown it to have bette compromise in stability.

Comparison testing was performed on a cluster with significant message loads, using a variety of filesystem creat monitored was the "Request Local Time", indicating the amount of time append operations were taking. XFS result EXT4 configuration), as well as lower average wait times. The XFS performance also showed less variability in disl

General Filesystem Notes

For any filesystem used for data directories, on Linux systems, the following options are recommended to be used

- noatime: This option disables updating of a file's atime (last access time) attribute when the file is read. This ca the case of bootstrapping consumers. Kafka does not rely on the atime attributes at all, so it is safe to disable t

XFS Notes

The XFS filesystem has a significant amount of auto-tuning in place, so it does not require any change in the defau tuning parameters worth considering are:

- largeio: This affects the preferred I/O size reported by the stat call. While this can allow for higher performance performance.
- nobarrier: For underlying devices that have battery-backed cache, this option can provide a little more performa device is well-behaved, it will report to the filesystem that it does not require flushes, and this option will have n

EXT4 Notes

EXT4 is a serviceable choice of filesystem for the Kafka data directories, however getting the most performance o these options are generally unsafe in a failure scenario, and will result in much more data loss and corruption. For can be wiped and the replicas rebuilt from the cluster. In a multiple-failure scenario, such as a power outage, this c is not easily recoverable. The following options can be adjusted:

- data=writeback: Ext4 defaults to data=ordered which puts a strong order on some writes. Kafka does not requi unflushed log. This setting removes the ordering constraint and seems to significantly reduce latency.
- Disabling journaling: Journaling is a tradeoff: it makes reboots faster after server crashes but it introduces a gr performance. Those who don't care about reboot time and want to reduce a major source of write latency spike
- commit=num_secs: This tunes the frequency with which ext4 commits to its metadata journal. Setting this to a Setting this to a higher value will improve throughput.
- nobh: This setting controls additional ordering guarantees when using data=writeback mode. This should be sa throughput and latency.
- delalloc: Delayed allocation means that the filesystem avoid allocating any blocks until the physical write occur pages and helps ensure the data is written sequentially. This feature is great for throughput. It does seem to inv variance.

6.6 Monitoring

Kafka uses Yammer Metrics for metrics reporting in the server. The Java clients use Kafka Metrics, a built-in metri client applications. Both expose metrics via JMX and can be configured to report stats using pluggable stats repo

All Kafka rate metrics have a corresponding cumulative count metric with suffix `-total` . For example, `recor` `records-consumed-total` .

The easiest way to see the available metrics is to fire up jconsole and point it at a running kafka client or server; th

Security Considerations for Remote Monitoring using JMX

Apache Kafka disables remote JMX by default. You can enable remote monitoring using JMX by setting the enviro
or standard Java system properties to enable remote JMX programmatically. You must enable security when enab
unauthorized users cannot monitor or control your broker or application as well as the platform on which these are
Kafka and security configs must be overridden for production deployments by setting the environment variable K
setting appropriate Java system properties. See [Monitoring and Management Using JMX Technology]() for details o

We do graphing and alerting on the following metrics:

DESCRIPTION	MBEAN NAME	NORMAL VALUE
Message in rate	kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec	
Byte in rate from clients	kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec	
Byte in rate from other brokers	kafka.server:type=BrokerTopicMetrics,name=ReplicationBytesInPerSec	
Request rate	kafka.network:type=RequestMetrics,name=RequestsPerSec,request={Produce|FetchConsumer|FetchFollower}	
Error rate	kafka.network:type=RequestMetrics,name=ErrorsPerSec,request=([-.\w]+),error=([-.\w]+)	Number of errors in responses coun request-type, per-error-code. If a res contains multiple errors, all are cour ror=NONE indicates successful resp
Request size in bytes	kafka.network:type=RequestMetrics,name=RequestBytes,request=([-.\w]+)	Size of requests for each request ty
Temporary memory size in bytes	kafka.network:type=RequestMetrics,name=TemporaryMemoryBytes,request={Produce|Fetch}	Temporary memory used for messa mat conversions and decompressio
Message conversion time	kafka.network:type=RequestMetrics,name=MessageConversionsTimeMs,request={Produce|Fetch}	Time in milliseconds spent on mess mat conversions.
Message conversion rate	kafka.server:type=BrokerTopicMetrics,name={Produce|Fetch}MessageConversionsPerSec,topic=([-.\w]+)	Number of records which required n format conversion.
Byte out rate to clients	kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec	
Byte out rate to other brokers	kafka.server:type=BrokerTopicMetrics,name=ReplicationBytesOutPerSec	
Message validation failure rate due to no key specified for compacted topic	kafka.server:type=BrokerTopicMetrics,name=NoKeyCompactedTopicRecordsPerSec	
Message validation failure rate due to invalid magic number	kafka.server:type=BrokerTopicMetrics,name=InvalidMagicNumberRecordsPerSec	
Message validation failure rate due to incorrect crc checksum	kafka.server:type=BrokerTopicMetrics,name=InvalidMessageCrcRecordsPerSec	
Message validation failure rate due to non-continuous offset or sequence number in batch	kafka.server:type=BrokerTopicMetrics,name=InvalidOffsetOrSequenceRecordsPerSec	
Log flush rate and time	kafka.log:type=LogFlushStats,name=LogFlushRateAndTimeMs	
# of under replicated partitions (|ISR| < |all replicas|)	kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions	0
# of under minIsr partitions (|ISR| < min.insync.replicas)	kafka.server:type=ReplicaManager,name=UnderMinIsrPartitionCount	0
# of at minIsr partitions (|ISR| = min.insync.replicas)	kafka.server:type=ReplicaManager,name=AtMinIsrPartitionCount	0
# of offline log directories	kafka.log:type=LogManager,name=OfflineLogDirectoryCount	0

Is controller active on broker	kafka.controller:type=KafkaController,name=ActiveControllerCount	only one broker in the cluster shoulc
Leader election rate	kafka.controller:type=ControllerStats,name=LeaderElectionRateAndTimeMs	non-zero when there are broker failu
Unclean leader election rate	kafka.controller:type=ControllerStats,name=UncleanLeaderElectionsPerSec	0
Pending topic deletes	kafka.controller:type=KafkaController,name=TopicsToDeleteCount	
Pending replica deletes	kafka.controller:type=KafkaController,name=ReplicasToDeleteCount	
Ineligible pending topic deletes	kafka.controller:type=KafkaController,name=TopicsIneligibleToDeleteCount	
Ineligible pending replica deletes	kafka.controller:type=KafkaController,name=ReplicasIneligibleToDeleteCount	
Partition counts	kafka.server:type=ReplicaManager,name=PartitionCount	mostly even across brokers
Leader replica counts	kafka.server:type=ReplicaManager,name=LeaderCount	mostly even across brokers
ISR shrink rate	kafka.server:type=ReplicaManager,name=IsrShrinksPerSec	If a broker goes down, ISR for some partitions will shrink. When that brol again, ISR will be expanded once the are fully caught up. Other than that, pected value for both ISR shrink rate pansion rate is 0.
ISR expansion rate	kafka.server:type=ReplicaManager,name=IsrExpandsPerSec	See above
Max lag in messages btw follower and leader replicas	kafka.server:type=ReplicaFetcherManager,name=MaxLag,clientId=Replica	lag should be proportional to the ma batch size of a produce request.
Lag in messages per follower replica	kafka.server:type=FetcherLagMetrics,name=ConsumerLag,clientId=([-.\w]+),topic=([-.\w]+),partition=([0-9]+)	lag should be proportional to the ma batch size of a produce request.
Requests waiting in the producer purgatory	kafka.server:type=DelayedOperationPurgatory,name=PurgatorySize,delayedOperation=Produce	non-zero if ack=-1 is used
Requests waiting in the fetch purgatory	kafka.server:type=DelayedOperationPurgatory,name=PurgatorySize,delayedOperation=Fetch	size depends on fetch.wait.max.ms consumer
Request total time	kafka.network:type=RequestMetrics,name=TotalTimeMs,request={Produce|FetchConsumer|FetchFollower}	broken into queue, local, remote anc sponse send time
Time the request waits in the request queue	kafka.network:type=RequestMetrics,name=RequestQueueTimeMs,request={Produce|FetchConsumer|FetchFollower}	
Time the request is processed at the leader	kafka.network:type=RequestMetrics,name=LocalTimeMs,request={Produce|FetchConsumer|FetchFollower}	
Time the request waits for the follower	kafka.network:type=RequestMetrics,name=RemoteTimeMs,request={Produce|FetchConsumer|FetchFollower}	non-zero for produce requests wher
Time the request waits in the response queue	kafka.network:type=RequestMetrics,name=ResponseQueueTimeMs,request={Produce|FetchConsumer|FetchFollower}	
Time to send the response	kafka.network:type=RequestMetrics,name=ResponseSendTimeMs,request={Produce|FetchConsumer|FetchFollower}	
Number of messages the consumer lags behind the producer by. Published by the consumer, not broker.	kafka.consumer:type=consumer-fetch-manager-metrics,client-id={client-id} Attribute: records-lag-max	
The average fraction of time the network processors are idle	kafka.network:type=SocketServer,name=NetworkProcessorAvgIdlePercent	between 0 and 1, ideally > 0.3

The number of connections disconnected on a processor due to a client not re-authenticating and then using the connection beyond its expiration time for anything other than re-authentication	kafka.server:type=socket-server-metrics,listener=[SASL_PLAINTEXT|SASL_SSL],networkProcessor=<#>,name=expired-connections-killed-count	ideally 0 when re-authentication is e implying there are no longer any old 2.2.0 clients connecting to this (liste cessor) combination
The total number of connections disconnected, across all processors, due to a client not re-authenticating and then using the connection beyond its expiration time for anything other than re-authentication	kafka.network:type=SocketServer,name=ExpiredConnectionsKilledCount	ideally 0 when re-authentication is e implying there are no longer any old 2.2.0 clients connecting to this brok
The average fraction of time the request handler threads are idle	kafka.server:type=KafkaRequestHandlerPool,name=RequestHandlerAvgIdlePercent	between 0 and 1, ideally > 0.3
Bandwidth quota metrics per (user, client-id), user or client-id	kafka.server:type={Produce|Fetch},user=([-.\w]+),client-id=([-.\w]+)	Two attributes. throttle-time indicate amount of time in ms the client was tled. Ideally = 0. byte-rate indicates t produce/consume rate of the client bytes/sec. For (user, client-id) quota user and client-id are specified. If pe quota is applied to the client, user is specified. If per-user quota is applie id is not specified.
Request quota metrics per (user, client-id), user or client-id	kafka.server:type=Request,user=([-.\w]+),client-id=([-.\w]+)	Two attributes. throttle-time indicate amount of time in ms the client was tled. Ideally = 0. request-time indicat percentage of time spent in broker r and I/O threads to process requests client group. For (user, client-id) quo user and client-id are specified. If pe quota is applied to the client, user is specified. If per-user quota is applie id is not specified.
Requests exempt from throttling	kafka.server:type=Request	exempt-throttle-time indicates the p age of time spent in broker network threads to process requests that are from throttling.
ZooKeeper client request latency	kafka.server:type=ZooKeeperClientMetrics,name=ZooKeeperRequestLatencyMs	Latency in millseconds for ZooKeep quests from broker.
ZooKeeper connection status	kafka.server:type=SessionExpireListener,name=SessionState	Connection status of broker's ZooKe session which may be one of Disconnected|SyncConnected|AuthF nnectedReadOnly|SaslAuthenticatec
Max time to load group metadata	kafka.server:type=group-coordinator-metrics,name=partition-load-time-max	maximum time, in milliseconds, it to load offsets and group metadata fro consumer offset partitions loaded in 30 seconds
Avg time to load group metadata	kafka.server:type=group-coordinator-metrics,name=partition-load-time-avg	average time, in milliseconds, it took offsets and group metadata from th sumer offset partitions loaded in the seconds
Max time to load transaction metadata	kafka.server:type=transaction-coordinator-metrics,name=partition-load-time-max	maximum time, in milliseconds, it to load transaction metadata from the sumer offset partitions loaded in the seconds
Avg time to load transaction metadata	kafka.server:type=transaction-coordinator-metrics,name=partition-load-time-avg	average time, in milliseconds, it took transaction metadata from the cons set partitions loaded in the last 30 s

Common monitoring metrics for producer/consumer/connect/streams

The following metrics are available on producer/consumer/connector/streams instances. For specific metrics, ple

METRIC/ATTRIBUTE NAME	DESCRIPTION	MBEAN NAME
connection-close-rate	Connections closed per second in the	kafka.[producer|consumer|connect]

	window.	[producer|consumer|connect]-metric id=([-.\w]+)
connection-close-total	Total connections closed in the window.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
connection-creation-rate	New connections established per second in the window.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
connection-creation-total	Total new connections established in the window.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
network-io-rate	The average number of network operations (reads or writes) on all connections per second.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
network-io-total	The total number of network operations (reads or writes) on all connections.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
outgoing-byte-rate	The average number of outgoing bytes sent per second to all servers.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
outgoing-byte-total	The total number of outgoing bytes sent to all servers.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
request-rate	The average number of requests sent per second.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
request-total	The total number of requests sent.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
request-size-avg	The average size of all requests in the window.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
request-size-max	The maximum size of any request sent in the window.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
incoming-byte-rate	Bytes/second read off all sockets.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
incoming-byte-total	Total bytes read off all sockets.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
response-rate	Responses received per second.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
response-total	Total responses received.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
select-rate	Number of times the I/O layer checked for new I/O to perform per second.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
select-total	Total number of times the I/O layer checked for new I/O to perform.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
io-wait-time-ns-avg	The average length of time the I/O thread spent waiting for a socket ready for reads or writes in nanoseconds.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
io-wait-ratio	The fraction of time the I/O thread spent waiting.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)
io-time-ns-avg	The average length of time for I/O per select call in nanoseconds.	kafka.[producer|consumer|connect] [producer|consumer|connect]-metric id=([-.\w]+)

io-ratio	The fraction of time the I/O thread spent doing I/O.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
connection-count	The current number of active connections.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
successful-authentication-rate	Connections per second that were successfully authenticated using SASL or SSL.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
successful-authentication-total	Total connections that were successfully authenticated using SASL or SSL.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
failed-authentication-rate	Connections per second that failed authentication.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
failed-authentication-total	Total connections that failed authentication.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
successful-reauthentication-rate	Connections per second that were successfully re-authenticated using SASL.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
successful-reauthentication-total	Total connections that were successfully re-authenticated using SASL.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
reauthentication-latency-max	The maximum latency in ms observed due to re-authentication.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
reauthentication-latency-avg	The average latency in ms observed due to re-authentication.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
failed-reauthentication-rate	Connections per second that failed re-authentication.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
failed-reauthentication-total	Total connections that failed re-authentication.	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
successful-authentication-no-reauth-total	Total connections that were successfully authenticated by older, pre-2.2.0 SASL clients that do not support re-authentication. May only be non-zero	kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)

Common Per-broker metrics for producer/consumer/connect/streams

The following metrics are available on producer/consumer/connector/streams instances. For specific metrics, ple

METRIC/ATTRIBUTE NAME	DESCRIPTION	MBEAN NAME
outgoing-byte-rate	The average number of outgoing bytes sent per second for a node.	kafka.[producer|consumer|connect]:type=[consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([
outgoing-byte-total	The total number of outgoing bytes sent for a node.	kafka.[producer|consumer|connect]:type=[consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([
request-rate	The average number of requests sent per second for a node.	kafka.[producer|consumer|connect]:type=[consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([
request-total	The total number of requests sent for a node.	kafka.[producer|consumer|connect]:type=[consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([
request-size-avg	The average size of all requests in the win-	kafka.[producer|consumer|connect]

	dow for a node.	[consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([
request-size-max	The maximum size of any request sent in the window for a node.	kafka.[producer|consumer|connect] [consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([
incoming-byte-rate	The average number of bytes received per second for a node.	kafka.[producer|consumer|connect] [consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([
incoming-byte-total	The total number of bytes received for a node.	kafka.[producer|consumer|connect] [consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([
request-latency-avg	The average request latency in ms for a node.	kafka.[producer|consumer|connect] [consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([
request-latency-max	The maximum request latency in ms for a node.	kafka.[producer|consumer|connect] [consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([
response-rate	Responses received per second for a node.	kafka.[producer|consumer|connect] [consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([
response-total	Total responses received for a node.	kafka.[producer|consumer|connect] [consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([

Producer monitoring

The following metrics are available on producer instances.

METRIC/ATTRIBUTE NAME	DESCRIPTION	MBEAN NAME
waiting-threads	The number of user threads blocked waiting for buffer memory to enqueue their records.	kafka.producer:type=producer-metri id=([-.\w]+)
buffer-total-bytes	The maximum amount of buffer memory the client can use (whether or not it is currently used).	kafka.producer:type=producer-metri id=([-.\w]+)
buffer-available-bytes	The total amount of buffer memory that is not being used (either unallocated or in the free list).	kafka.producer:type=producer-metri id=([-.\w]+)
bufferpool-wait-time	The fraction of time an appender waits for space allocation.	kafka.producer:type=producer-metri id=([-.\w]+)

Producer Sender Metrics

kafka.producer:type=producer-metrics,client-id="{client-id}"		
	ATTRIBUTE NAME	DESCRIPTION
	batch-size-avg	The average number of bytes sent p tion per-request.
	batch-size-max	The max number of bytes sent per p per-request.
	batch-split-rate	The average number of batch splits second
	batch-split-total	The total number of batch splits
	compression-rate-avg	The average compression rate of re batches.
	metadata-age	The age in seconds of the current pr metadata being used.

	produce-throttle-time-avg	The average time in ms a request wa tled by a broker
	produce-throttle-time-max	The maximum time in ms a request throttled by a broker
	record-error-rate	The average per-second number of r sends that resulted in errors
	record-error-total	The total number of record sends th ed in errors
	record-queue-time-avg	The average time in ms record batch in the send buffer.
	record-queue-time-max	The maximum time in ms record ba spent in the send buffer.
	record-retry-rate	The average per-second number of r record sends
	record-retry-total	The total number of retried record s
	record-send-rate	The average number of records sent second.
	record-send-total	The total number of records sent.
	record-size-avg	The average record size
	record-size-max	The maximum record size
	records-per-request-avg	The average number of records per
	request-latency-avg	The average request latency in ms
	request-latency-max	The maximum request latency in ms
	requests-in-flight	The current number of in-flight requ awaiting a response.
kafka.producer:type=producer-topic-metrics,client-id="{client-id}",topic="{topic}"		
	ATTRIBUTE NAME	DESCRIPTION
	byte-rate	The average number of bytes sent p ond for a topic.
	byte-total	The total number of bytes sent for a
	compression-rate	The average compression rate of re batches for a topic.
	record-error-rate	The average per-second number of r sends that resulted in errors for a to
	record-error-total	The total number of record sends th ed in errors for a topic
	record-retry-rate	The average per-second number of r record sends for a topic
	record-retry-total	The total number of retried record s a topic
	record-send-rate	The average number of records sent ond for a topic.
	record-send-total	The total number of records sent for

consumer monitoring

The following metrics are available on consumer instances.

METRIC/ATTRIBUTE NAME	DESCRIPTION	MBEAN NAME
time-between-poll-avg	The average delay between invocations of poll().	kafka.consumer:type=consumer-metrics,client-id=([-.\w]+)

time-between-poll-max	The max delay between invocations of poll().	kafka.consumer:type=consumer-metrics,client-id=([-.\w]+)
last-poll-seconds-ago	The number of seconds since the last poll() invocation.	kafka.consumer:type=consumer-metrics,client-id=([-.\w]+)
poll-idle-ratio-avg	The average fraction of time the consumer's poll() is idle as opposed to waiting for the user code to process records.	kafka.consumer:type=consumer-metrics,client-id=([-.\w]+)

Consumer Group Metrics

METRIC/ATTRIBUTE NAME	DESCRIPTION	MBEAN NAME
commit-latency-avg	The average time taken for a commit request	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
commit-latency-max	The max time taken for a commit request	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
commit-rate	The number of commit calls per second	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
commit-total	The total number of commit calls	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
assigned-partitions	The number of partitions currently assigned to this consumer	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
heartbeat-response-time-max	The max time taken to receive a response to a heartbeat request	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
heartbeat-rate	The average number of heartbeats per second	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
heartbeat-total	The total number of heartbeats	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
join-time-avg	The average time taken for a group rejoin	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
join-time-max	The max time taken for a group rejoin	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
join-rate	The number of group joins per second	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
join-total	The total number of group joins	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
sync-time-avg	The average time taken for a group sync	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
sync-time-max	The max time taken for a group sync	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
sync-rate	The number of group syncs per second	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
sync-total	The total number of group syncs	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
rebalance-latency-avg	The average time taken for a group rebalance	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
rebalance-latency-max	The max time taken for a group rebalance	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
rebalance-latency-total	The total time taken for group rebalances so far	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
rebalance-total	The total number of group rebalances participated	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
rebalance-rate-per-hour	The number of group rebalance participated per hour	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
failed-rebalance-total	The total number of failed group rebalances	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)

failed-rebalance-rate-per-hour	The number of failed group rebalance event per hour	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
last-rebalance-seconds-ago	The number of seconds since the last rebalance event	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
last-heartbeat-seconds-ago	The number of seconds since the last controller heartbeat	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
partitions-revoked-latency-avg	The average time taken by the on-partitions-revoked rebalance listener callback	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
partitions-revoked-latency-max	The max time taken by the on-partitions-revoked rebalance listener callback	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
partitions-assigned-latency-avg	The average time taken by the on-partitions-assigned rebalance listener callback	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
partitions-assigned-latency-max	The max time taken by the on-partitions-assigned rebalance listener callback	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
partitions-lost-latency-avg	The average time taken by the on-partitions-lost rebalance listener callback	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)
partitions-lost-latency-max	The max time taken by the on-partitions-lost rebalance listener callback	kafka.consumer:type=consumer-co metrics,client-id=([-.\w]+)

Consumer Fetch Metrics

kafka.consumer:type=consumer-fetch-manager-metrics,client-id="{client-id}"		
	ATTRIBUTE NAME	DESCRIPTION
	bytes-consumed-rate	The average number of bytes consu second
	bytes-consumed-total	The total number of bytes consume
	fetch-latency-avg	The average time taken for a fetch r
	fetch-latency-max	The max time taken for any fetch re
	fetch-rate	The number of fetch requests per se
	fetch-size-avg	The average number of bytes fetche request
	fetch-size-max	The maximum number of bytes fetc request
	fetch-throttle-time-avg	The average throttle time in ms
	fetch-throttle-time-max	The maximum throttle time in ms
	fetch-total	The total number of fetch requests.
	records-consumed-rate	The average number of records con per second
	records-consumed-total	The total number of records consun
	records-lag-max	The maximum lag in terms of numb records for any partition in this wind
	records-lead-min	The minimum lead in terms of numb records for any partition in this wind
	records-per-request-avg	The average number of records in e request
kafka.consumer:type=consumer-fetch-manager-metrics,client-id="{client-id}",topic="{topic}"		
	ATTRIBUTE NAME	DESCRIPTION
	bytes-consumed-rate	The average number of bytes consu second for a topic
	bytes-consumed-total	The total number of bytes consume topic

	ATTRIBUTE NAME	DESCRIPTION
	fetch-size-avg	The average number of bytes fetche quest for a topic
	fetch-size-max	The maximum number of bytes fetc request for a topic
	records-consumed-rate	The average number of records con per second for a topic
	records-consumed-total	The total number of records consun topic
	records-per-request-avg	The average number of records in ea quest for a topic
kafka.consumer:type=consumer-fetch-manager-metrics,partition="{partition}",topic="{topic}",client-id="{client-id}"		
	ATTRIBUTE NAME	DESCRIPTION
	preferred-read-replica	The current read replica for the parti if reading from leader
	records-lag	The latest lag of the partition
	records-lag-avg	The average lag of the partition
	records-lag-max	The max lag of the partition
	records-lead	The latest lead of the partition
	records-lead-avg	The average lead of the partition
	records-lead-min	The min lead of the partition

Connect Monitoring

A Connect worker process contains all the producer and consumer metrics as well as metrics specific to Connect. connector and task have additional metrics.

kafka.connect:type=connect-worker-metrics		
	ATTRIBUTE NAME	DESCRIPTION
	connector-count	The number of connectors run in thi
	connector-startup-attempts-total	The total number of connector start this worker has attempted.
	connector-startup-failure-percentage	The average percentage of this worl nectors starts that failed.
	connector-startup-failure-total	The total number of connector start failed.
	connector-startup-success-percentage	The average percentage of this worl nectors starts that succeeded.
	connector-startup-success-total	The total number of connector start succeeded.
	task-count	The number of tasks run in this worl
	task-startup-attempts-total	The total number of task startups th worker has attempted.
	task-startup-failure-percentage	The average percentage of this worl tasks starts that failed.
	task-startup-failure-total	The total number of task starts that
	task-startup-success-percentage	The average percentage of this worl tasks starts that succeeded.
	task-startup-success-total	The total number of task starts that succeeded.
kafka.connect:type=connect-worker-metrics,connector="{connector}"		

	ATTRIBUTE NAME	DESCRIPTION
	connector-destroyed-task-count	The number of destroyed tasks of th nector on the worker.
	connector-failed-task-count	The number of failed tasks of the co on the worker.
	connector-paused-task-count	The number of paused tasks of the tor on the worker.
	connector-running-task-count	The number of running tasks of the tor on the worker.
	connector-total-task-count	The number of tasks of the connect worker.
	connector-unassigned-task-count	The number of unassigned tasks of nector on the worker.

kafka.connect:type=connect-worker-rebalance-metrics

	ATTRIBUTE NAME	DESCRIPTION
	completed-rebalances-total	The total number of rebalances com by this worker.
	connect-protocol	The Connect protocol used by this c
	epoch	The epoch or generation number of worker.
	leader-name	The name of the group leader.
	rebalance-avg-time-ms	The average time in milliseconds sp this worker to rebalance.
	rebalance-max-time-ms	The maximum time in milliseconds this worker to rebalance.
	rebalancing	Whether this worker is currently reb
	time-since-last-rebalance-ms	The time in milliseconds since this v completed the most recent rebalanc

kafka.connect:type=connector-metrics,connector="{connector}"

	ATTRIBUTE NAME	DESCRIPTION
	connector-class	The name of the connector class.
	connector-type	The type of the connector. One of 'so 'sink'.
	connector-version	The version of the connector class, ed by the connector.
	status	The status of the connector. One of signed', 'running', 'paused', 'failed', or 'destroyed'.

kafka.connect:type=connector-task-metrics,connector="{connector}",task="{task}"

	ATTRIBUTE NAME	DESCRIPTION
	batch-size-avg	The average size of the batches pro by the connector.
	batch-size-max	The maximum size of the batches p by the connector.
	offset-commit-avg-time-ms	The average time in milliseconds tal this task to commit offsets.
	offset-commit-failure-percentage	The average percentage of this task commit attempts that failed.
	offset-commit-max-time-ms	The maximum time in milliseconds this task to commit offsets.
	offset-commit-success-percentage	The average percentage of this task commit attempts that succeeded.

	pause-ratio	The fraction of time this task has sp pause state.
	running-ratio	The fraction of time this task has sp running state.
	status	The status of the connector task. Or 'unassigned', 'running', 'paused', 'faile 'destroyed'.
kafka.connect:type=sink-task-metrics,connector="{connector}",task="{task}"		
	ATTRIBUTE NAME	DESCRIPTION
	offset-commit-completion-rate	The average per-second number of commit completions that were com successfully.
	offset-commit-completion-total	The total number of offset commit c tions that were completed successf
	offset-commit-seq-no	The current sequence number for of commits.
	offset-commit-skip-rate	The average per-second number of commit completions that were recei late and skipped/ignored.
	offset-commit-skip-total	The total number of offset commit c tions that were received too late and skipped/ignored.
	partition-count	The number of topic partitions assig this task belonging to the named sir nector in this worker.
	put-batch-avg-time-ms	The average time taken by this task batch of sinks records.
	put-batch-max-time-ms	The maximum time taken by this tas a batch of sinks records.
	sink-record-active-count	The number of records that have be from Kafka but not yet completely c ted/flushed/acknowledged by the si
	sink-record-active-count-avg	The average number of records that been read from Kafka but not yet co committed/flushed/acknowledged b sink task.
	sink-record-active-count-max	The maximum number of records th been read from Kafka but not yet co committed/flushed/acknowledged b sink task.
	sink-record-lag-max	The maximum lag in terms of numb records that the sink task is behind sumer's position for any topic partiti
	sink-record-read-rate	The average per-second number of read from Kafka for this task belong the named sink connector in this wo is before transformations are applie
	sink-record-read-total	The total number of records read fro by this task belonging to the named nector in this worker, since the task restarted.
	sink-record-send-rate	The average per-second number of output from the transformations and sent/put to this task belonging to th sink connector in this worker. This is transformations are applied and exc any records filtered out by the transformations.
	sink-record-send-total	The total number of records output transformations and sent/put to this

		longing to the named sink connecto worker, since the task was last resta
kafka.connect:type=source-task-metrics,connector="{connector}",task="{task}"		
	ATTRIBUTE NAME	DESCRIPTION
	poll-batch-avg-time-ms	The average time in milliseconds tal this task to poll for a batch of sourc records.
	poll-batch-max-time-ms	The maximum time in milliseconds this task to poll for a batch of sourc records.
	source-record-active-count	The number of records that have be duced by this task but not yet comp written to Kafka.
	source-record-active-count-avg	The average number of records that been produced by this task but not y pletely written to Kafka.
	source-record-active-count-max	The maximum number of records th been produced by this task but not y pletely written to Kafka.
	source-record-poll-rate	The average per-second number of produced/polled (before transforma this task belonging to the named so nector in this worker.
	source-record-poll-total	The total number of records produc (before transformation) by this task ing to the named source connector i worker.
	source-record-write-rate	The average per-second number of output from the transformations an to Kafka for this task belonging to th source connector in this worker. Thi transformations are applied and exc any records filtered out by the transformations.
	source-record-write-total	The number of records output from formations and written to Kafka for belonging to the named source con this worker, since the task was last r
kafka.connect:type=task-error-metrics,connector="{connector}",task="{task}"		
	ATTRIBUTE NAME	DESCRIPTION
	deadletterqueue-produce-failures	The number of failed writes to the d queue.
	deadletterqueue-produce-requests	The number of attempted writes to t letter queue.
	last-error-timestamp	The epoch timestamp when this tas countered an error.
	total-errors-logged	The number of errors that were logg
	total-record-errors	The number of record processing er this task.
	total-record-failures	The number of record processing fa this task.
	total-records-skipped	The number of records skipped due
	total-retries	The number of operations retried.

Streams Monitoring

A Kafka Streams instance contains all the producer and consumer metrics as well as additional metrics specific t
recording levels: debug and info. The debug level records all metrics, while the info level records only the thread-le

Note that the metrics have a 4-layer hierarchy. At the top level there are client-level metrics for each started Kafka
metrics. Each stream thread has tasks, with their own metrics. Each task has a number of processor nodes, with t
record caches, all with their own metrics.

Use the following configuration option to specify which metrics you want collected:

```
metrics.recording.level="info"
```

Client Metrics

All the following metrics have a recording level of `info` :

METRIC/ATTRIBUTE NAME	DESCRIPTION	MBEAN NAME
version	The version of the Kafka Streams client.	kafka.streams:type=stream-metrics ([-.\w]+)
commit-id	The version control commit ID of the Kafka Streams client.	kafka.streams:type=stream-metrics ([-.\w]+)
application-id	The application ID of the Kafka Streams client.	kafka.streams:type=stream-metrics ([-.\w]+)
topology-description	The description of the topology executed in the Kafka Streams client.	kafka.streams:type=stream-metrics ([-.\w]+)
state	The state of the Kafka Streams client.	kafka.streams:type=stream-metrics ([-.\w]+)

Thread Metrics

All the following metrics have a recording level of `info` :

METRIC/ATTRIBUTE NAME	DESCRIPTION	MBEAN NAME
commit-latency-avg	The average execution time in ms for committing, across all running tasks of this thread.	kafka.streams:type=stream-metrics ([-.\w]+)
commit-latency-max	The maximum execution time in ms for committing across all running tasks of this thread.	kafka.streams:type=stream-metrics ([-.\w]+)
poll-latency-avg	The average execution time in ms for polling, across all running tasks of this thread.	kafka.streams:type=stream-metrics ([-.\w]+)
poll-latency-max	The maximum execution time in ms for polling across all running tasks of this thread.	kafka.streams:type=stream-metrics ([-.\w]+)
process-latency-avg	The average execution time in ms for processing, across all running tasks of this thread.	kafka.streams:type=stream-metrics ([-.\w]+)
process-latency-max	The maximum execution time in ms for processing across all running tasks of this thread.	kafka.streams:type=stream-metrics ([-.\w]+)
punctuate-latency-avg	The average execution time in ms for punc-	kafka.streams:type=stream-metrics

	tuating, across all running tasks of this thread.	([-.\w]+)
punctuate-latency-max	The maximum execution time in ms for punctuating across all running tasks of this thread.	kafka.streams:type=stream-metrics ([-.\w]+)
commit-rate	The average number of commits per second.	kafka.streams:type=stream-metrics ([-.\w]+)
commit-total	The total number of commit calls across all tasks.	kafka.streams:type=stream-metrics ([-.\w]+)
poll-rate	The average number of polls per second.	kafka.streams:type=stream-metrics ([-.\w]+)
poll-total	The total number of poll calls across all tasks.	kafka.streams:type=stream-metrics ([-.\w]+)
process-rate	The average number of process calls per second.	kafka.streams:type=stream-metrics ([-.\w]+)
process-total	The total number of process calls across all tasks.	kafka.streams:type=stream-metrics ([-.\w]+)
punctuate-rate	The average number of punctuates per second.	kafka.streams:type=stream-metrics ([-.\w]+)
punctuate-total	The total number of punctuate calls across all tasks.	kafka.streams:type=stream-metrics ([-.\w]+)
task-created-rate	The average number of newly created tasks per second.	kafka.streams:type=stream-metrics ([-.\w]+)
task-created-total	The total number of tasks created.	kafka.streams:type=stream-metrics ([-.\w]+)
task-closed-rate	The average number of tasks closed per second.	kafka.streams:type=stream-metrics ([-.\w]+)
task-closed-total	The total number of tasks closed.	kafka.streams:type=stream-metrics ([-.\w]+)
skipped-records-rate	The average number of skipped records per second.	kafka.streams:type=stream-metrics ([-.\w]+)
skipped-records-total	The total number of skipped records.	kafka.streams:type=stream-metrics ([-.\w]+)

Task Metrics

All the following metrics have a recording level of `debug` :

METRIC/ATTRIBUTE NAME	DESCRIPTION	MBEAN NAME
commit-latency-avg	The average commit time in ns for this task.	kafka.streams:type=stream-task-metrics,client-id=([-.\w]+),task-id=([-.
commit-latency-max	The maximum commit time in ns for this task.	kafka.streams:type=stream-task-metrics,client-id=([-.\w]+),task-id=([-.
commit-rate	The average number of commit calls per second.	kafka.streams:type=stream-task-metrics,client-id=([-.\w]+),task-id=([-.
commit-total	The total number of commit calls.	kafka.streams:type=stream-task-metrics,client-id=([-.\w]+),task-id=([-.
record-lateness-avg	The average observed lateness of records.	kafka.streams:type=stream-task-metrics,client-id=([-.\w]+),task-id=([-.
record-lateness-max	The max observed lateness of records.	kafka.streams:type=stream-task-metrics,client-id=([-.\w]+),task-id=([-.

Processor Node Metrics

All the following metrics have a recording level of `debug`:

METRIC/ATTRIBUTE NAME	DESCRIPTION	MBEAN NAME
process-latency-avg	The average process execution time in ns.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
process-latency-max	The maximum process execution time in ns.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
punctuate-latency-avg	The average punctuate execution time in ns.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
punctuate-latency-max	The maximum punctuate execution time in ns.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
create-latency-avg	The average create execution time in ns.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
create-latency-max	The maximum create execution time in ns.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
destroy-latency-avg	The average destroy execution time in ns.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
destroy-latency-max	The maximum destroy execution time in ns.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
process-rate	The average number of process operations per second.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
process-total	The total number of process operations called.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
punctuate-rate	The average number of punctuate operations per second.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
punctuate-total	The total number of punctuate operations called.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
create-rate	The average number of create operations per second.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
create-total	The total number of create operations called.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
destroy-rate	The average number of destroy operations per second.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
destroy-total	The total number of destroy operations called.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
forward-rate	The average rate of records being forwarded downstream, from source nodes only, per second.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
forward-total	The total number of of records being forwarded downstream, from source nodes only.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)
suppression-emit-rate	The rate at which records that have been emitted downstream from suppression operation nodes. Compare with the `process-`	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)

	`rate` metric to determine how many updates are being suppressed.	
suppression-emit-total	The total number of records that have been emitted downstream from suppression operation nodes. Compare with the `process-total` metric to determine how many updates are being suppressed.	kafka.streams:type=stream-process metrics,client-id=([-.\w]+),task-id=([-.\w]+),processor-node-id=([-.\w]+)

State Store Metrics

All the following metrics have a recording level of `debug`. Note that the `store-scope` value is specified in S stores; for built-in state stores, currently we have:

- `in-memory-state`
- `in-memory-lru-state`
- `in-memory-window-state`
- `rocksdb-state` (for RocksDB backed key-value store)
- `rocksdb-window-state` (for RocksDB backed window store)
- `rocksdb-session-state` (for RocksDB backed session store)

METRIC/ATTRIBUTE NAME	DESCRIPTION	MBEAN NAME
put-latency-avg	The average put execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
put-latency-max	The maximum put execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
put-if-absent-latency-avg	The average put-if-absent execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
put-if-absent-latency-max	The maximum put-if-absent execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
get-latency-avg	The average get execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
get-latency-max	The maximum get execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
delete-latency-avg	The average delete execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
delete-latency-max	The maximum delete execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
put-all-latency-avg	The average put-all execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
put-all-latency-max	The maximum put-all execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
all-latency-avg	The average all operation execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
all-latency-max	The maximum all operation execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)

range-latency-avg	The average range execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
range-latency-max	The maximum range execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
flush-latency-avg	The average flush execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
flush-latency-max	The maximum flush execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
restore-latency-avg	The average restore execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
restore-latency-max	The maximum restore execution time in ns.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
put-rate	The average put rate for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
put-total	The total number of put calls for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
put-if-absent-rate	The average put-if-absent rate for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
put-if-absent-total	The total number of put-if-absent calls for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
get-rate	The average get rate for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
get-total	The total number of get calls for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
delete-rate	The average delete rate for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
delete-total	The total number of delete calls for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
put-all-rate	The average put-all rate for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
put-all-total	The total number of put-all calls for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
all-rate	The average all operation rate for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
all-total	The total number of all operation calls for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
range-rate	The average range rate for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
range-total	The total number of range calls for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
flush-rate	The average flush rate for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-.

		[store-scope]-id=([-.\w]+)
flush-total	The total number of flush calls for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
restore-rate	The average restore rate for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
restore-total	The total number of restore calls for this store.	kafka.streams:type=stream-[store-s metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)

RocksDB Metrics

All the following metrics have a recording level of `debug`. The metrics are collected every minute from the Rock instances as it is the case for aggregations over time and session windows, each metric reports an aggregation o `store-scope` for built-in RocksDB state stores are currently the following:

- `rocksdb-state` (for RocksDB backed key-value store)
- `rocksdb-window-state` (for RocksDB backed window store)
- `rocksdb-session-state` (for RocksDB backed session store)

METRIC/ATTRIBUTE NAME	DESCRIPTION	MBEAN NAME
bytes-written-rate	The average number of bytes written per second to the RocksDB state store.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
bytes-written-total	The total number of bytes written to the RocksDB state store.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
bytes-read-rate	The average number of bytes read per second from the RocksDB state store.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
bytes-read-total	The total number of bytes read from the RocksDB state store.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
memtable-bytes-flushed-rate	The average number of bytes flushed per second from the memtable to disk.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
memtable-bytes-flushed-total	The total number of bytes flushed from the memtable to disk.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
memtable-hit-ratio	The ratio of memtable hits relative to all lookups to the memtable.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
block-cache-data-hit-ratio	The ratio of block cache hits for data blocks relative to all lookups for data blocks to the block cache.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
block-cache-index-hit-ratio	The ratio of block cache hits for index blocks relative to all lookups for index blocks to the block cache.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
block-cache-filter-hit-ratio	The ratio of block cache hits for filter blocks relative to all lookups for filter blocks to the block cache.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
write-stall-duration-avg	The average duration of write stalls in ms.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-. [store-scope]-id=([-.\w]+)
write-stall-duration-total	The total duration of write stalls in ms.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-.

		[store-scope]-id=([-.\w]+)
bytes-read-compaction-rate	The average number of bytes read per second during compaction.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-.\w]+),[store-scope]-id=([-.\w]+)
bytes-written-compaction-rate	The average number of bytes written per second during compaction.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-.\w]+),[store-scope]-id=([-.\w]+)
number-open-files	The number of current open files.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-.\w]+),[store-scope]-id=([-.\w]+)
number-file-errors-total	The total number of file errors occurred.	kafka.streams:type=stream-state-metrics,client-id=([-.\w]+),task-id=([-.\w]+),[store-scope]-id=([-.\w]+)

Record Cache Metrics

All the following metrics have a recording level of `debug` :

METRIC/ATTRIBUTE NAME	DESCRIPTION	MBEAN NAME
hitRatio-avg	The average cache hit ratio defined as the ratio of cache read hits over the total cache read requests.	kafka.streams:type=stream-record-cache-metrics,client-id=([-.\w]+),task-id=([-.\w]+),record-cache-id=([-.\w]+)
hitRatio-min	The mininum cache hit ratio.	kafka.streams:type=stream-record-cache-metrics,client-id=([-.\w]+),task-id=([-.\w]+),record-cache-id=([-.\w]+)
hitRatio-max	The maximum cache hit ratio.	kafka.streams:type=stream-record-cache-metrics,client-id=([-.\w]+),task-id=([-.\w]+),record-cache-id=([-.\w]+)

Suppression Buffer Metrics

All the following metrics have a recording level of `debug` :

suppression-buffer-size-current	The current total size, in bytes, of the buffered data.	kafka.streams:type=stream-buffer-metrics,client-id=([-.\w]+),task-id=([-.\w]+),buffer-id=([-.\w]+)
suppression-buffer-size-avg	The average total size, in bytes, of the buffered data over the sampling window.	kafka.streams:type=stream-buffer-metrics,client-id=([-.\w]+),task-id=([-.\w]+),buffer-id=([-.\w]+)
suppression-buffer-size-max	The maximum total size, in bytes, of the buffered data over the sampling window.	kafka.streams:type=stream-buffer-metrics,client-id=([-.\w]+),task-id=([-.\w]+),buffer-id=([-.\w]+)
suppression-buffer-count-current	The current number of records buffered.	kafka.streams:type=stream-buffer-metrics,client-id=([-.\w]+),task-id=([-.\w]+),buffer-id=([-.\w]+)
suppression-buffer-size-avg	The average number of records buffered over the sampling window.	kafka.streams:type=stream-buffer-metrics,client-id=([-.\w]+),task-id=([-.\w]+),buffer-id=([-.\w]+)
suppression-buffer-size-max	The maximum number of records buffered over the sampling window.	kafka.streams:type=stream-buffer-metrics,client-id=([-.\w]+),task-id=([-.\w]+),buffer-id=([-.\w]+)

Others

We recommend monitoring GC time and other stats and various server stats such as CPU utilization, I/O service ti message/byte rate (global and per topic), request rate/size/time, and on the consumer side, max lag in messages keep up, max lag needs to be less than a threshold and min fetch rate needs to be larger than 0.

6.7 ZooKeeper

Stable version

The current stable branch is 3.5. Kafka is regularly updated to include the latest release in the 3.5 series.

Operationalizing ZooKeeper

Operationally, we do the following for a healthy ZooKeeper installation:

- Redundancy in the physical/hardware/network layout: try not to put them all in the same rack, decent (but don't paths, etc. A typical ZooKeeper ensemble has 5 or 7 servers, which tolerates 2 and 3 servers down, respectively acceptable, but keep in mind that you'll only be able to tolerate 1 server down in this case.
- I/O segregation: if you do a lot of write type traffic you'll almost definitely want the transaction logs on a dedica batched for performance), and consequently, concurrent writes can significantly affect performance. ZooKeepe ideally should be written on a disk group separate from the transaction log. Snapshots are written to disk asyno and message log files. You can configure a server to use a separate disk group with the dataLogDir parameter.
- Application segregation: Unless you really understand the application patterns of other apps that you want to ir isolation (though this can be a balancing act with the capabilities of the hardware).
- Use care with virtualization: It can work, depending on your cluster layout and read/write patterns and SLAs, bu up and throw off ZooKeeper, as it can be very time sensitive
- ZooKeeper configuration: It's java, make sure you give it 'enough' heap space (We usually run them with 3-5G, bu Unfortunately we don't have a good formula for it, but keep in mind that allowing for more ZooKeeper state mea recovery time. In fact, if the snapshot becomes too large (a few gigabytes), then you may need to increase the i join the ensemble.
- Monitoring: Both JMX and the 4 letter words (4lw) commands are very useful, they do overlap in some cases (a more predictable, or at the very least, they work better with the LI monitoring infrastructure)
- Don't overbuild the cluster: large clusters, especially in a write heavy usage pattern, means a lot of intracluster c member updates), but don't underbuild it (and risk swamping the cluster). Having more servers adds to your rea

Overall, we try to keep the ZooKeeper system as small as will handle the load (plus standard growth capacity plan with the configuration or application layout as compared to the official release as well as keep it as self contained versions, since it has a tendency to try to put things in the OS standard hierarchy, which can be 'messy', for want of

7. SECURITY

7.1 Security Overview

In release 0.9.0.0, the Kafka community added a number of features that, used either separately or together, increa are currently supported:

1. Authentication of connections to brokers from clients (producers and consumers), other brokers and tools, us mechanisms:
   - SASL/GSSAPI (Kerberos) - starting at version 0.9.0.0
   - SASL/PLAIN - starting at version 0.10.0.0
   - SASL/SCRAM-SHA-256 and SASL/SCRAM-SHA-512 - starting at version 0.10.2.0
   - SASL/OAUTHBEARER - starting at version 2.0
2. Authentication of connections from brokers to ZooKeeper
3. Encryption of data transferred between brokers and clients, between brokers, or between brokers and tools us enabled, the magnitude of which depends on the CPU type and the JVM implementation.)
4. Authorization of read / write operations by clients
5. Authorization is pluggable and integration with external authorization services is supported

It's worth noting that security is optional - non-secured clusters are supported, as well as a mix of authenticated, u below explain how to configure and use the security features in both clients and brokers.

7.2 Encryption and Authentication using SSL

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

1. **Generate SSL key and certificate for each Kafka broker**

   The first step of deploying one or more brokers with the SSL support is to generate the key and the certificate to accomplish this task. We will generate the key into a temporary keystore initially so that we can export and

   ```
   keytool -keystore server.keystore.jks -alias localhost -validity {validity} -genke
   ```

   You need to specify two parameters in the above command:
   1. keystore: the keystore file that stores the certificate. The keystore file contains the private key of the certi
   2. validity: the valid time of the certificate in days.

   #### Configuring Host Name Verification

   From Kafka version 2.0.0 onwards, host name verification of servers is enabled by default for client connectic middle attacks. Server host name verification may be disabled by setting `ssl.endpoint.identificatic`

   ```
   ssl.endpoint.identification.algorithm=
   ```

   For dynamically configured broker listeners, hostname verification may be disabled using `kafka-configs.`

   ```
   bin/kafka-configs.sh --bootstrap-server localhost:9093 --entity-type brokers --en
   ```

   For older versions of Kafka, `ssl.endpoint.identification.algorithm` is not defined by default, so set to `HTTPS` to enable host name verification.

   ```
   ssl.endpoint.identification.algorithm=HTTPS
   ```

   Host name verification must be enabled to prevent man-in-the-middle attacks if server endpoints are not valid

Configuring Host Name In Certificates

If host name verification is enabled, clients will verify the server's fully qualified domain name (FQDN) against

1. Common Name (CN)
2. Subject Alternative Name (SAN)

Both fields are valid, RFC-2818 recommends the use of SAN however. SAN is also more flexible, allowing for r
CN can be set to a more meaningful value for authorization purposes. To add a SAN field append the followin

```
keytool -keystore server.keystore.jks -alias localhost -validity {validity} -genke
```

The following command can be run afterwards to verify the contents of the generated certificate:

```
keytool -list -v -keystore server.keystore.jks
```

2. Creating your own CA

After the first step, each machine in the cluster has a public-private key pair, and a certificate to identify the m
attacker can create such a certificate to pretend to be any machine.

Therefore, it is important to prevent forged certificates by signing them for each machine in the cluster. A cert
works likes a government that issues passports—the government stamps (signs) each passport so that the p
stamps to ensure the passport is authentic. Similarly, the CA signs the certificates, and the cryptography guar
Thus, as long as the CA is a genuine and trusted authority, the clients have high assurance that they are conn

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

The generated CA is simply a public-private key pair and certificate, and it is intended to sign other certificate
The next step is to add the generated CA to the **clients' truststore** so that the clients can trust this CA:

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

Note: If you configure the Kafka brokers to require client authentication by setting ssl.client.auth to be "reques
provide a truststore for the Kafka brokers as well and it should have all the CA certificates that clients' keys w

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

In contrast to the keystore in step 1 that stores each machine's own identity, the truststore of a client stores a
into one's truststore also means trusting all certificates that are signed by that certificate. As the analogy abo
(certificates) that it has issued. This attribute is called the chain of trust, and it is particularly useful when dep
the cluster with a single CA, and have all machines share the same truststore that trusts the CA. That way all

3. Signing the certificate

The next step is to sign all certificates generated by step 1 with the CA generated in step 2. First, you need to

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

Then sign it with the CA:

```
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days
```

Finally, you need to 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 parameters are the following:

1. keystore: the location of the keystore
2. ca-cert: the certificate of the CA
3. ca-key: the private key of the CA
4. ca-password: the passphrase of the CA
5. cert-file: the exported, unsigned certificate of the server
6. cert-signed: the signed certificate of the server

Here is an example of a bash script with all above steps. Note that one of the commands assumes a passwo
before running it.

```
#!/bin/bash
#Step 1
keytool -keystore server.keystore.jks -alias localhost -validity
#Step 2
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
keytool -keystore server.truststore.jks -alias CARoot -import -fi
keytool -keystore client.truststore.jks -alias CARoot -import -fi
#Step 3
keytool -keystore server.keystore.jks -alias localhost -certreq -
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out ce
keytool -keystore server.keystore.jks -alias CARoot -import -file
keytool -keystore server.keystore.jks -alias localhost -import -f
```

4. Configuring Kafka Brokers

Kafka Brokers support listening for connections on multiple ports. We need to configure the following propert
separated values:

```
listeners
```

If SSL is not enabled for inter-broker communication (see below for how to enable it), both PLAINTEXT and S

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

Following SSL configs are needed on the broker side

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

Note: ssl.truststore.password is technically optional but highly recommended. If a password is not set access disabled. Optional settings that are worth considering:

1. ssl.client.auth=none ("required" => client authentication is required, "requested" => client authentication is "requested" is discouraged as it provides a false sense of security and misconfigured clients will still con
2. ssl.cipher.suites (Optional). A cipher suite is a named combination of authentication, encryption, MAC an for a network connection using TLS or SSL network protocol. (Default is an empty list)
3. ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1 (list out the SSL protocols that you are going to accept fro SSL in production is not recommended)
4. ssl.keystore.type=JKS
5. ssl.truststore.type=JKS
6. ssl.secure.random.implementation=SHA1PRNG

If you want to enable SSL for inter-broker communication, add the following to the server.properties file (it def

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

Due to import regulations in some countries, the Oracle implementation limits the strength of cryptographic a (for example, AES with 256-bit keys), the [JCE Unlimited Strength Jurisdiction Policy Files] must be obtained ar for more information.

The JRE/JDK will have a default pseudo-random number generator (PRNG) that is used for cryptography oper with the

```
ssl.secure.random.implementation
```

. However, there are performance issues with some implementations (notably, the default chosen on Linux sy

```
NativePRNG
```

, utilizes a global lock). In cases where performance of SSL connections becomes an issue, consider explicitly

```
SHA1PRNG
```

implementation is non-blocking, and has shown very good performance characteristics under heavy load (50

Once you start the broker you should be able to see in the server.log

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

To check quickly if the server keystore and truststore are setup properly you can 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 server's certificate:

```
            -----BEGIN CERTIFICATE-----
            {variable sized random bytes}
            -----END CERTIFICATE-----
            subject=/C=US/ST=CA/L=Santa Clara/O=org/OU=org/CN=Sriharsha Chint
            issuer=/C=US/ST=CA/L=Santa Clara/O=org/OU=org/CN=kafka/emailAddre
```

If the certificate does not show up or if there are any other error messages then your keystore is not setup pro

5. **Configuring Kafka Clients**

SSL is supported only for the new Kafka Producer and Consumer, the older API is not supported. The configs
If client authentication is not required in the broker, then the following is a minimal configuration example:

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

Note: ssl.truststore.password is technically optional but highly recommended. If a password is not set access
disabled. If client authentication is required, then a keystore must be created like in step 1 and the following n

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

Other configuration settings that may also be needed depending on our requirements and the broker configur

1. ssl.provider (Optional). The name of the security provider used for SSL connections. Default value is the
2. ssl.cipher.suites (Optional). A cipher suite is a named combination of authentication, encryption, MAC an for a network connection using TLS or SSL network protocol.
3. ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1. It should list at least one of the protocols configured on t
4. ssl.truststore.type=JKS
5. ssl.keystore.type=JKS

Examples using console-producer and console-consumer:

```
kafka-console-producer.sh --broker-list localhost:9093 --topic test --producer.co
kafka-console-consumer.sh --bootstrap-server localhost:9093 --topic test --consum
```

7.3 Authentication using SASL

1. **JAAS configuration**

Kafka uses the Java Authentication and Authorization Service (JAAS) for SASL configuration.

1. **JAAS configuration for Kafka brokers**

   `KafkaServer` is the section name in the JAAS file used by each KafkaServer/Broker. This section provides client connections made by the broker for inter-broker communication. If multiple listeners are configure name in lower-case followed by a period, e.g. `sasl_ssl.KafkaServer`.

   `Client` section is used to authenticate a SASL connection with zookeeper. It also allows the brokers to se that only the brokers can modify it. It is necessary to have the same principal name across all brokers. If property `zookeeper.sasl.clientconfig` to the appropriate name (*e.g.*, `-Dzookeeper.sasl.clientconfig=Zk`

   ZooKeeper uses "zookeeper" as the service name by default. If you want to change this, set the system p (*e.g.*, `-Dzookeeper.sasl.client.username=zk`).

   Brokers may also configure JAAS using the broker configuration property `sasl.jaas.config`. The p SASL mechanism, i.e. `listener.name.{listenerName}.{saslMechanism}.sasl.jaas.conf` multiple mechanisms are configured on a listener, configs must be provided for each mechanism using t

   ```
   listener.name.sasl_ssl.scram-sha-256.sasl.jaas.config=org.apache.kafka.common.
       username="admin" \
       password="admin-secret";
   listener.name.sasl_ssl.plain.sasl.jaas.config=org.apache.kafka.common.security
       username="admin" \
       password="admin-secret" \
       user_admin="admin-secret" \
       user_alice="alice-secret";
   ```

   If JAAS configuration is defined at different levels, the order of precedence used is:

   - Broker configuration property `listener.name.{listenerName}.{saslMechanism}.sasl.ja`
   - `{listenerName}.KafkaServer` section of static JAAS configuration
   - `KafkaServer` section of static JAAS configuration

   Note that ZooKeeper JAAS config may only be configured using static JAAS configuration.

   See GSSAPI (Kerberos), PLAIN, SCRAM or OAUTHBEARER for example broker configurations.

2. **JAAS configuration for Kafka clients**

   Clients may configure JAAS using the client configuration property sasl.jaas.config or using the static JA

   1. **JAAS configuration using client configuration property**

      Clients may specify JAAS configuration as a producer or consumer property without creating a phys and consumers within the same JVM to use different credentials by specifying different properties f `java.security.auth.login.config` and client property `sasl.jaas.config` are specifi

      See GSSAPI (Kerberos), PLAIN, SCRAM or OAUTHBEARER for example configurations.

2. **JAAS configuration using static config file**

To configure SASL authentication on the clients using static JAAS config file:

1. Add a JAAS config file with a client login section named `KafkaClient`. Configure a login module examples for setting up GSSAPI (Kerberos), PLAIN, SCRAM or OAUTHBEARER. For example, GS

```
    KafkaClient {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    keyTab="/etc/security/keytabs/kafka_client.keytab"
    principal="kafka-client-1@EXAMPLE.COM";
};
```

2. Pass the JAAS config file location as JVM parameter to each client JVM. For example:

```
-Djava.security.auth.login.config=/etc/kafka/kafka_client_jaas.conf
```

2. SASL configuration

SASL may be used with PLAINTEXT or SSL as the transport layer using the security protocol SASL_PLAINTEX also be configured.

1. SASL mechanisms

Kafka supports the following SASL mechanisms:

- GSSAPI (Kerberos)
- PLAIN
- SCRAM-SHA-256
- SCRAM-SHA-512
- OAUTHBEARER

2. SASL configuration for Kafka brokers

1. Configure a SASL port in server.properties, by adding at least one of SASL_PLAINTEXT or SASL_SSL separated values:

```
listeners=SASL_PLAINTEXT://host.name:port
```

If you are only configuring a SASL port (or if you want the Kafka brokers to authenticate each other u inter-broker communication:

```
security.inter.broker.protocol=SASL_PLAINTEXT (or SASL_SSL)
```

2. Select one or more supported mechanisms to enable in the broker and follow the steps to configure broker, follow the steps here.

3. **SASL configuration for Kafka clients**

SASL authentication is only supported for the new Java Kafka producer and consumer, the older API is n

To configure SASL authentication on the clients, select a SASL mechanism that is enabled in the broker f the selected mechanism.

3. Authentication using SASL/Kerberos

1. Prerequisites

1. **Kerberos**
   If your organization is already using a Kerberos server (for example, by using Active Directory), there need to install one, your Linux vendor likely has packages for Kerberos and a short guide on how to i Oracle Java, you will need to download JCE policy files for your Java version and copy them to $JAV
2. **Create Kerberos Principals**
   If you are using the organization's Kerberos or Active Directory server, ask your Kerberos administrat operating system user that will access Kafka with Kerberos authentication (via clients and tools).
   If you have installed your own Kerberos, you will need to create these principals yourself using the fc

```
sudo /usr/sbin/kadmin.local -q 'addprinc -randkey kafka/{hostname}@{REALM}
sudo /usr/sbin/kadmin.local -q "ktadd -k /etc/security/keytabs/{keytabname
```

3. **Make sure all hosts can be reachable using hostnames** - it is a Kerberos requirement that all your h

2. Configuring Kafka Brokers

1. Add a suitably modified JAAS file similar to the one below to each Kafka broker's config directory, le broker should have its own keytab):

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

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

```
16   };
```

`KafkaServer` section in the JAAS file tells the broker which principal to use and the location of the ke using the keytab specified in this section. See notes for more details on Zookeeper SASL configurati

2. Pass the JAAS and optionally the krb5 file locations as JVM parameters to each Kafka broker (see h

```
-Djava.security.krb5.conf=/etc/kafka/krb5.conf
    -Djava.security.auth.login.config=/etc/kafka/kafka_server_jaas
```

3. Make sure the keytabs configured in the JAAS file are readable by the operating system user who is
4. Configure SASL port and SASL mechanisms in server.properties as described here. For example:

```
listeners=SASL_PLAINTEXT://host.name:port
    security.inter.broker.protocol=SASL_PLAINTEXT
    sasl.mechanism.inter.broker.protocol=GSSAPI
    sasl.enabled.mechanisms=GSSAPI
```

We must also configure the service name in server.properties, which should match the principal nan "kafka/kafka1.hostname.com@EXAMPLE.com", so:

```
sasl.kerberos.service.name=kafka
```

3. **Configuring Kafka Clients**

To configure SASL authentication on the clients:

1. Clients (producers, consumers, connect workers, etc) will authenticate to the cluster with their own p client), so obtain or create these principals as needed. Then configure the JAAS configuration prope different users by specifying different principals. The property `sasl.jaas.config` in producer.p producer and consumer can connect to the Kafka Broker. The following is an example configuration processes):

```
sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule requi
    useKeyTab=true \
    storeKey=true  \
    keyTab="/etc/security/keytabs/kafka_client.keytab" \
    principal="kafka-client-1@EXAMPLE.COM";
```

For command-line utilities like kafka-console-consumer or kafka-console-producer, kinit can be usec

```
sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule requ
    useTicketCache=true;
```

JAAS configuration for clients may alternatively be specified as a JVM parameter similar to brokers `KafkaClient`. This option allows only one user for all client connections from a JVM.

2. Make sure the keytabs configured in the JAAS configuration are readable by the operating system u
3. Optionally pass the krb5 file locations as JVM parameters to each client JVM (see here for more de

```
-Djava.security.krb5.conf=/etc/kafka/krb5.conf
```

4. Configure the following properties in producer.properties or consumer.properties:

```
security.protocol=SASL_PLAINTEXT (or SASL_SSL)
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
```

4. Authentication using SASL/PLAIN

SASL/PLAIN is a simple username/password authentication mechanism that is typically used with TLS for er default implementation for SASL/PLAIN which can be extended for production use as described here.

The username is used as the authenticated `Principal` for configuration of ACLs etc.

1. Configuring Kafka Brokers

1. Add a suitably modified JAAS file similar to the one below to each Kafka broker's config directory, le

```
KafkaServer {
    org.apache.kafka.common.security.plain.PlainLoginModule required
    username="admin"
    password="admin-secret"
    user_admin="admin-secret"
    user_alice="alice-secret";
};
```

This configuration defines two users (*admin* and *alice*). The properties `username` and `password` in the to other brokers. In this example, *admin* is the user for inter-broker communication. The set of prope connect to the broker and the broker validates all client connections including those from other brok

2. Pass the JAAS config file location as JVM parameter to each Kafka broker:

```
-Djava.security.auth.login.config=/etc/kafka/kafka_server_jaas.con
```

3. Configure SASL port and SASL mechanisms in server.properties as described here. For example:

```
listeners=SASL_SSL://host.name:port
    security.inter.broker.protocol=SASL_SSL
    sasl.mechanism.inter.broker.protocol=PLAIN
    sasl.enabled.mechanisms=PLAIN
```

2. **Configuring Kafka Clients**

   To configure SASL authentication on the clients:

   1. Configure the JAAS configuration property for each client in producer.properties or consumer.prope consumer can connect to the Kafka Broker. The following is an example configuration for a client fo

      ```
      sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule r
          username="alice" \
          password="alice-secret";
      ```

      The options `username` and `password` are used by clients to configure the user for client connections. I clients within a JVM may connect as different users by specifying different user names and passwo

      JAAS configuration for clients may alternatively be specified as a JVM parameter similar to brokers `KafkaClient`. This option allows only one user for all client connections from a JVM.

   2. Configure the following properties in producer.properties or consumer.properties:

      ```
      security.protocol=SASL_SSL
      sasl.mechanism=PLAIN
      ```

3. **Use of SASL/PLAIN in production**

   - SASL/PLAIN should be used only with SSL as transport layer to ensure that clear passwords are not t
   - The default implementation of SASL/PLAIN in Kafka specifies usernames and passwords in the JAAS you can avoid storing clear passwords on disk by configuring your own callback handlers that obtain configuration options `sasl.server.callback.handler.class` and `sasl.client.callba`
   - In production systems, external authentication servers may implement password authentication. Fron handlers that use external authentication servers for password verification by configuring `sasl.ser`

5. **Authentication using SASL/SCRAM**

Salted Challenge Response Authentication Mechanism (SCRAM) is a family of SASL mechanisms that addres perform username/password authentication like PLAIN and DIGEST-MD5. The mechanism is defined in RFC 5 can be used with TLS to perform secure authentication. The username is used as the authenticated `Princi` implementation in Kafka stores SCRAM credentials in Zookeeper and is suitable for use in Kafka installations Considerations for more details.

1. **Creating SCRAM Credentials**

   The SCRAM implementation in Kafka uses Zookeeper as credential store. Credentials can be created in Z
   enabled, credentials must be created by adding a config with the mechanism name. Credentials for inter-
   started. Client credentials may be created and updated dynamically and updated credentials will be used

   Create SCRAM credentials for user *alice* with password *alice-secret*:

   ```
   > bin/kafka-configs.sh --zookeeper localhost:2181 --alter --add-config 'SCRAM-
   ```

   The default iteration count of 4096 is used if iterations are not specified. A random salt is created and th
   ServerKey are stored in Zookeeper. See [RFC 5802](#) for details on SCRAM identity and the individual fields.

   The following examples also require a user *admin* for inter-broker communication which can be created u

   ```
   > bin/kafka-configs.sh --zookeeper localhost:2181 --alter --add-config 'SCRAM-
   ```

   Existing credentials may be listed using the *--describe* option:

   ```
   > bin/kafka-configs.sh --zookeeper localhost:2181 --describe --entity-type use
   ```

   Credentials may be deleted for one or more SCRAM mechanisms using the *--delete* option:

   ```
   > bin/kafka-configs.sh --zookeeper localhost:2181 --alter --delete-config 'SCF
   ```

2. **Configuring Kafka Brokers**

   1. Add a suitably modified JAAS file similar to the one below to each Kafka broker's config directory, le

      ```
      KafkaServer {
          org.apache.kafka.common.security.scram.ScramLoginModule require
          username="admin"
          password="admin-secret";
      };
      ```

      The properties `username` and `password` in the `KafkaServer` section are used by the broker to initiate co
      inter-broker communication.
   2. Pass the JAAS config file location as JVM parameter to each Kafka broker:

      ```
      -Djava.security.auth.login.config=/etc/kafka/kafka_server_jaas.con
      ```

   3. Configure SASL port and SASL mechanisms in server.properties as described [here](#). For example:

      ```
      listeners=SASL_SSL://host.name:port
      security.inter.broker.protocol=SASL_SSL
      ```

```
sasl.mechanism.inter.broker.protocol=SCRAM-SHA-256 (or SCRAM-SHA-5:
sasl.enabled.mechanisms=SCRAM-SHA-256 (or SCRAM-SHA-512)
```

3. **Configuring Kafka Clients**

To configure SASL authentication on the clients:

1. Configure the JAAS configuration property for each client in producer.properties or consumer.prope consumer can connect to the Kafka Broker. The following is an example configuration for a client fo

```
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule r
     username="alice" \
     password="alice-secret";
```

The options `username` and `password` are used by clients to configure the user for client connections. I clients within a JVM may connect as different users by specifying different user names and passwo

JAAS configuration for clients may alternatively be specified as a JVM parameter similar to brokers `KafkaClient`. This option allows only one user for all client connections from a JVM.

2. Configure the following properties in producer.properties or consumer.properties:

```
security.protocol=SASL_SSL
sasl.mechanism=SCRAM-SHA-256 (or SCRAM-SHA-512)
```

4. **Security Considerations for SASL/SCRAM**

- The default implementation of SASL/SCRAM in Kafka stores SCRAM credentials in Zookeeper. This is secure and on a private network.
- Kafka supports only the strong hash functions SHA-256 and SHA-512 with a minimum iteration count and high iteration counts protect against brute force attacks if Zookeeper security is compromised.
- SCRAM should be used only with TLS-encryption to prevent interception of SCRAM exchanges. This p impersonation if Zookeeper is compromised.
- From Kafka version 2.0 onwards, the default SASL/SCRAM credential store may be overridden using c `sasl.server.callback.handler.class` in installations where Zookeeper is not secure.
- For more details on security considerations, refer to RFC 5802.

6. Authentication using SASL/OAUTHBEARER

The OAuth 2 Authorization Framework "enables a third-party application to obtain limited access to an HTTP : approval interaction between the resource owner and the HTTP service, or by allowing the third-party applicat mechanism enables the use of the framework in a SASL (i.e. a non-HTTP) context; it is defined in RFC 7628. T validates Unsecured JSON Web Tokens and is only suitable for use in non-production Kafka installations. Refe

1. **Configuring Kafka Brokers**

    1. Add a suitably modified JAAS file similar to the one below to each Kafka broker's config directory, le

    ```
    KafkaServer {
        org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginMo
        unsecuredLoginStringClaim_sub="admin";
    };
    ```

    The property `unsecuredLoginStringClaim_sub` in the `KafkaServer` section is used by the broker when appear in the subject (`sub`) claim and will be the user for inter-broker communication.

    2. Pass the JAAS config file location as JVM parameter to each Kafka broker:

    ```
    -Djava.security.auth.login.config=/etc/kafka/kafka_server_jaas.con
    ```

    3. Configure SASL port and SASL mechanisms in server.properties as described [here](). For example:

    ```
    listeners=SASL_SSL://host.name:port (or SASL_PLAINTEXT if non-produ
    security.inter.broker.protocol=SASL_SSL (or SASL_PLAINTEXT if non-p
    sasl.mechanism.inter.broker.protocol=OAUTHBEARER
    sasl.enabled.mechanisms=OAUTHBEARER
    ```

2. **Configuring Kafka Clients**

    To configure SASL authentication on the clients:

    1. Configure the JAAS configuration property for each client in producer.properties or consumer.prope consumer can connect to the Kafka Broker. The following is an example configuration for a client fo

    ```
    sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerL
        unsecuredLoginStringClaim_sub="alice";
    ```

    The option `unsecuredLoginStringClaim_sub` is used by clients to configure the subject (`sub`) claim, w clients connect to the broker as user *alice*. Different clients within a JVM may connect as different u `sasl.jaas.config` .

    JAAS configuration for clients may alternatively be specified as a JVM parameter similar to brokers `KafkaClient`. This option allows only one user for all client connections from a JVM.

    2. Configure the following properties in producer.properties or consumer.properties:

    ```
    security.protocol=SASL_SSL (or SASL_PLAINTEXT if non-production)
    sasl.mechanism=OAUTHBEARER
    ```

3. The default implementation of SASL/OAUTHBEARER depends on the jackson-databind library. Since dependency via their build tool.

3. **Unsecured Token Creation Options for SASL/OAUTHBEARER**

- The default implementation of SASL/OAUTHBEARER in Kafka creates and validates Unsecured JSON provide the flexibility to create arbitrary tokens in a DEV or TEST environment.
- Here are the various supported JAAS module options on the client side (and on the broker side if OAU

JAAS Module Option for Unsecured Token Creation	Documentation
`unsecuredLoginStringClaim_<claimname>="value"`	Creates a `String` claim with the given name and value. Any valid claim name can be specified except '`iat`' and '`exp`' (these are automatically generated).
`unsecuredLoginNumberClaim_<claimname>="value"`	Creates a `Number` claim with the given name and value. Any valid claim name can be specified except '`iat`' and '`exp`' (these are automatically generated).
`unsecuredLoginListClaim_<claimname>="value"`	Creates a `String List` claim with the given name and values parsed from the given value where the first character is taken as the delimiter. For example: `unsecuredLoginListClaim_fubar="|value1|value2"`. Any valid claim name can be specified except '`iat`' and '`exp`' (these are automatically generated).
`unsecuredLoginExtension_<extensionname>="value"`	Creates a `String` extension with the given name and value. For example: `unsecuredLoginExtension_traceId="123"`. A valid extension name is any sequence of lowercase or uppercase alphabet characters. In addition, the "auth" extension name is reserved. A valid extension value is any combination of characters with ASCII codes 1-127.
`unsecuredLoginPrincipalClaimName`	Set to a custom claim name if you wish the name of the `String` claim holding the principal name to be something other than '`sub`'.
`unsecuredLoginLifetimeSeconds`	Set to an integer value if the token expiration is to be set to something other than the default value of 3600 seconds (which is 1 hour). The '`exp`' claim will be set to reflect the expiration time.
`unsecuredLoginScopeClaimName`	Set to a custom claim name if you wish the name of the `String` or `String List` claim holding any token scope to be something other than '`scope`'.

4. **Unsecured Token Validation Options for SASL/OAUTHBEARER**

- Here are the various supported JAAS module options on the broker side for Unsecured JSON Web Tol

JAAS Module Option for Unsecured Token Validation	Documentation
`unsecuredValidatorPrincipalClaimName="value"`	Set to a non-empty value if you wish a particular `String` claim holding a principal

	name to be checked for existence; the default is to check for the existence of the 'sub' claim.
`unsecuredValidatorScopeClaimName="value"`	Set to a custom claim name if you wish the name of the `String` or `String List` claim holding any token scope to be something other than 'scope'.
`unsecuredValidatorRequiredScope="value"`	Set to a space-delimited list of scope values if you wish the `String/String List` claim holding the token scope to be checked to make sure it contains certain values.
`unsecuredValidatorAllowableClockSkewMs="value"`	Set to a positive integer value if you wish to allow up to some number of positive milliseconds of clock skew (the default is 0).

- The default unsecured SASL/OAUTHBEARER implementation may be overridden (and must be overric Server callback handlers.
- For more details on security considerations, refer to [RFC 6749, Section 10](https://tools.ietf.org/html/rfc6749#section-10).

5. **Token Refresh for SASL/OAUTHBEARER**

Kafka periodically refreshes any token before it expires so that the client can continue to make connectic algorithm operates are specified as part of the producer/consumer/broker configuration and are as follo details. The default values are usually reasonable, in which case these configuration parameters would n

Producer/Consumer/Broker Configuration Property
`sasl.login.refresh.window.factor`
`sasl.login.refresh.window.jitter`
`sasl.login.refresh.min.period.seconds`
`sasl.login.refresh.min.buffer.seconds`

6. **Secure/Production Use of SASL/OAUTHBEARER**

Production use cases will require writing an implementation of `org.apache.kafka.common.security.auth` `org.apache.kafka.common.security.oauthbearer.OAuthBearerTokenCallback` and declaring it via either th non-broker client or via the `listener.name.sasl_ssl.oauthbearer.sasl.login.callback.handler.class` c inter-broker protocol).

Production use cases will also require writing an implementation of `org.apache.kafka.common.security.` `org.apache.kafka.common.security.oauthbearer.OAuthBearerValidatorCallback` and declaring it via the `listener.name.sasl_ssl.oauthbearer.sasl.server.callback.handler.class` broker configuration option.

7. **Security Considerations for SASL/OAUTHBEARER**

- The default implementation of SASL/OAUTHBEARER in Kafka creates and validates Unsecured JSON
- OAUTHBEARER should be used in production enviromnments only with TLS-encryption to prevent inte
- The default unsecured SASL/OAUTHBEARER implementation may be overridden (and must be overric
  Server callback handlers as described above.
- For more details on OAuth 2 security considerations in general, refer to RFC 6749, Section 10.

7. Enabling multiple SASL mechanisms in a broker

1. Specify configuration for the login modules of all enabled mechanisms in the `KafkaServer` section of the

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

            org.apache.kafka.common.security.plain.PlainLoginModule requir
            username="admin"
            password="admin-secret"
            user_admin="admin-secret"
            user_alice="alice-secret";
        };
```

2. Enable the SASL mechanisms in server.properties:

```
    sasl.enabled.mechanisms=GSSAPI,PLAIN,SCRAM-SHA-256,SCRAM-SHA-512,OAUTH
```

3. Specify the SASL security protocol and mechanism for inter-broker communication in server.properties i

```
    security.inter.broker.protocol=SASL_PLAINTEXT (or SASL_SSL)
    sasl.mechanism.inter.broker.protocol=GSSAPI (or one of the other enabl
```

4. Follow the mechanism-specific steps in GSSAPI (Kerberos), PLAIN, SCRAM and OAUTHBEARER to confi

8. Modifying SASL mechanism in a Running Cluster

SASL mechanism can be modified in a running cluster using the following sequence:

1. Enable new SASL mechanism by adding the mechanism to `sasl.enabled.mechanisms` in server.properties mechanisms as described [here](). Incrementally bounce the cluster nodes.
2. Restart clients using the new mechanism.
3. To change the mechanism of inter-broker communication (if this is required), set `sasl.mechanism.inter.` incrementally bounce the cluster again.
4. To remove old mechanism (if this is required), remove the old mechanism from `sasl.enabled.mechanisms` from JAAS config file. Incrementally bounce the cluster again.

9. ## Authentication using Delegation Tokens

Delegation token based authentication is a lightweight authentication mechanism to complement existing SA kafka brokers and clients. Delegation tokens will help processing frameworks to distribute the workload to av distributing Kerberos TGT/keytabs or keystores when 2-way SSL is used. See KIP-48 for more details.

Typical steps for delegation token usage are:

1. User authenticates with the Kafka cluster via SASL or SSL, and obtains a delegation token. This can be d
2. User securely passes the delegation token to Kafka clients for authenticating with the Kafka cluster.
3. Token owner/renewer can renew/expire the delegation tokens.

1. ### Token Management

   A master key/secret is used to generate and verify delegation tokens. This is supplied using config optio configured across all the brokers. If the secret is not set or set to empty string, brokers will disable the de

   In current implementation, token details are stored in Zookeeper and is suitable for use in Kafka installati master key/secret is stored as plain text in server.properties config file. We intend to make these configu

   A token has a current life, and a maximum renewable life. By default, tokens must be renewed once every `delegation.token.expiry.time.ms` and `delegation.token.max.lifetime.ms` config options.

   Tokens can also be cancelled explicitly. If a token is not renewed by the token's expiration time or if toker caches as well as from zookeeper.

2. ### Creating Delegation Tokens

   Tokens can be created by using Admin APIs or using `kafka-delegation-tokens.sh` script. Delegation toke on SASL or SSL authenticated channels. Tokens can not be requests if the initial authentication is done t examples are given below.

   Create a delegation token:

```
> bin/kafka-delegation-tokens.sh --bootstrap-server localhost:9092 --create
```

Renew a delegation token:

```
> bin/kafka-delegation-tokens.sh --bootstrap-server localhost:9092 --renew
```

Expire a delegation token:

```
> bin/kafka-delegation-tokens.sh --bootstrap-server localhost:9092 --expire
```

Existing tokens can be described using the --describe option:

```
> bin/kafka-delegation-tokens.sh --bootstrap-server localhost:9092 --describe
```

3. **Token Authentication**

Delegation token authentication piggybacks on the current SASL/SCRAM authentication mechanism. We
described in here.

Configuring Kafka Clients:

1. Configure the JAAS configuration property for each client in producer.properties or consumer.proper
consumer can connect to the Kafka Broker. The following is an example configuration for a client fo

```
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule r
     username="tokenID123" \
     password="lAYYSFmLs4bTjf+lTZ1LCHR/ZZFNA==" \
     tokenauth="true";
```

The options `username` and `password` are used by clients to configure the token id and token HMAC. A
authentication. In this example, clients connect to the broker using token id: *tokenID123*. Different cl
specifying different token details in `sasl.jaas.config`.

JAAS configuration for clients may alternatively be specified as a JVM parameter similar to brokers
`KafkaClient`. This option allows only one user for all client connections from a JVM.

4. **Procedure to manually rotate the secret:**

We require a re-deployment when the secret needs to be rotated. During this process, already connected
renew/expire requests with old tokens can fail. Steps are given below.

1. Expire all existing tokens.
2. Rotate the secret by rolling upgrade, and
3. Generate new tokens

We intend to automate this in a future Kafka release.

5. **Notes on Delegation Tokens**

- Currently, we only allow a user to create delegation token for that user only. Owner/Renewers can rene
own tokens. To describe others tokens, we need to add DESCRIBE permission on Token Resource.

7.4 Authorization and ACLs

Kafka ships with a pluggable Authorizer and an out-of-box authorizer implementation that uses zookeeper to store `authorizer.class.name` in server.properties. To enable the out of the box implementation use:

```
authorizer.class.name=kafka.security.auth.SimpleAclAuthorizer
```

Kafka acls are defined in the general format of "Principal P is [Allowed/Denied] Operation O From Host H on any R about the acl structure in KIP-11 and resource patterns in KIP-290. In order to add, remove or list acls you can use a specific Resource R, then R has no associated acls, and therefore no one other than super users is allowed to ac following in server.properties.

```
allow.everyone.if.no.acl.found=true
```

One can also add super users in server.properties like the following (note that the delimiter is semicolon since SSL "User" is case sensitive.

```
super.users=User:Bob;User:Alice
```

Customizing SSL User Name

By default, the SSL user name will be of the form "CN=writeuser,OU=Unknown,O=Unknown,L=Unknown,ST=Unknov `ssl.principal.mapping.rules` to a customized rule in server.properties. This config allows a list of rules evaluated in order and the first rule that matches a distinguished name is used to map it to a short name. Any late The format of `ssl.principal.mapping.rules` is a list where each rule starts with "RULE:" and contains an representation of the X.500 certificate distinguished name. If the distinguished name matches the pattern, then th supports lowercase/uppercase options, to force the translated result to be all lower/uppercase case. This is done

```
        RULE:pattern/replacement/
        RULE:pattern/replacement/[LU]
```

Example `ssl.principal.mapping.rules` values are:

```
        RULE:^CN=(.*?),OU=ServiceUsers.*$/$1/,
        RULE:^CN=(.*?),OU=(.*?),O=(.*?),L=(.*?),ST=(.*?),C=(.*?)$/$1@$2/L,
        RULE:^.*[Cc][Nn]=([a-zA-Z0-9.]*).*$/$1/L,
        DEFAULT
```

Above rules translate distinguished name "CN=serviceuser,OU=ServiceUsers,O=Unknown,L=Unknown,ST=Unknow
"CN=adminUser,OU=Admin,O=Unknown,L=Unknown,ST=Unknown,C=Unknown" to "adminuser@admin".
For advanced use cases, one can customize the name by setting a customized PrincipalBuilder in server.propertie

```
principal.builder.class=CustomizedPrincipalBuilderClass
```

Customizing SASL User Name

By default, the SASL user name will be the primary part of the Kerberos principal. One can change that by setting customized rule in server.properties. The format of `sasl.kerberos.principal.to.local.rules` is a list Kerberos configuration file (krb5.conf). This also support additional lowercase/uppercase rule, to force the transla "/L" or "/U" to the end of the rule. check below formats for syntax. Each rules starts with RULE: and contains an ex for more details.

```
        RULE:[n:string](regexp)s/pattern/replacement/
        RULE:[n:string](regexp)s/pattern/replacement/g
        RULE:[n:string](regexp)s/pattern/replacement//L
        RULE:[n:string](regexp)s/pattern/replacement/g/L
        RULE:[n:string](regexp)s/pattern/replacement//U
        RULE:[n:string](regexp)s/pattern/replacement/g/U
```

An example of adding a rule to properly translate user@MYDOMAIN.COM to user while also keeping the default ru

```
sasl.kerberos.principal.to.local.rules=RULE:[1:$1@$0](.*@MYDOMAIN.COM)s/@.*//,DEF
```

Command Line Interface

Kafka Authorization management CLI can be found under bin directory with all the other CLIs. The CLI script is cal supports:

OPTION	DESCRIPTION	DEFAULT
--add	Indicates to the script that user is trying to add an acl.	
--remove	Indicates to the script that user is trying to remove an acl.	
--list	Indicates to the script that user is trying to list acls.	
--authorizer	Fully qualified class name of the authorizer.	kafka.security.auth.SimpleAclAutho
--authorizer-properties	key=val pairs that will be passed to authorizer for initialization. For the default authorizer the example values are: zookeeper.connect=localhost:2181	

--bootstrap-server	A list of host/port pairs to use for establishing the connection to the Kafka cluster. Only one of --bootstrap-server or --authorizer option must be specified.	
--command-config	A property file containing configs to be passed to Admin Client. This option can only be used with --bootstrap-server option.	
--cluster	Indicates to the script that the user is trying to interact with acls on the singular cluster resource.	
--topic [topic-name]	Indicates to the script that the user is trying to interact with acls on topic resource pattern(s).	
--group [group-name]	Indicates to the script that the user is trying to interact with acls on consumer-group resource pattern(s)	
--transactional-id [transactional-id]	The transactionalId to which ACLs should be added or removed. A value of * indicates the ACLs should apply to all transactionalIds.	
--delegation-token [delegation-token]	Delegation token to which ACLs should be added or removed. A value of * indicates ACL should apply to all tokens.	
--resource-pattern-type [pattern-type]	Indicates to the script the type of resource pattern, (for --add), or resource pattern filter, (for --list and --remove), the user wishes to use. When adding acls, this should be a specific pattern type, e.g. 'literal' or 'prefixed'. When listing or removing acls, a specific pattern type filter can be used to list or remove acls from a specific type of resource pattern, or the filter values of 'any' or 'match' can be used, where 'any' will match any pattern type, but will match the resource name exactly, and 'match' will perform pattern matching to list or remove all acls that affect the supplied resource(s). WARNING: 'match', when used in combination with the '--remove' switch, should be used with care.	literal
--allow-principal	Principal is in PrincipalType:name format that will be added to ACL with Allow permission. Default PrincipalType string "User" is case sensitive. You can specify multiple --allow-principal in a single command.	
--deny-principal	Principal is in PrincipalType:name format that will be added to ACL with Deny permission. Default PrincipalType string "User" is case sensitive. You can specify multiple --deny-principal in a single command.	
--principal	Principal is in PrincipalType:name format that will be used along with --list option. Default PrincipalType string "User" is case sensitive. This will list the ACLs for the specified principal. You can specify multiple --principal in a single command.	
--allow-host	IP address from which principals listed in --allow-principal will have access.	if --allow-principal is specified defau which translates to "all hosts"
--deny-host	IP address from which principals listed in --deny-principal will be denied access.	if --deny-principal is specified defaul which translates to "all hosts"
--operation	Operation that will be allowed or denied. Valid values are: • Read	All

	• Write • Create • Delete • Alter • Describe • ClusterAction • DescribeConfigs • AlterConfigs • IdempotentWrite • All	
--producer	Convenience option to add/remove acls for producer role. This will generate acls that allows WRITE, DESCRIBE and CREATE on topic.	
--consumer	Convenience option to add/remove acls for consumer role. This will generate acls that allows READ, DESCRIBE on topic and READ on consumer-group.	
--idempotent	Enable idempotence for the producer. This should be used in combination with the --producer option. Note that idempotence is enabled automatically if the producer is authorized to a particular transactional-id.	
--force	Convenience option to assume yes to all queries and do not prompt.	

Examples

- **Adding Acls**
  Suppose you want to add an acl "Principals User:Bob and User:Alice are allowed to perform Operation Read anc
  198.51.100.1". You can do that by executing the CLI with following options:

  ```
  1   bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --add --
  ```

  By default, all principals that don't have an explicit acl that allows access for an operation to a resource are den
  to all but some principal we will have to use the --deny-principal and --deny-host option. For example, if we want
  User:BadBob from IP 198.51.100.3 we can do so using following commands:

  ```
  1   bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --add --
  ```

  Note that ``--allow-host`` and ``deny-host`` only support IP addresses (hostnames are not supported). Above ex
  resource pattern option. Similarly user can add acls to cluster by specifying --cluster and to a consumer group t
  resource of a certain type, e.g. suppose you wanted to add an acl "Principal User:Peter is allowed to produce to
  wildcard resource '*', e.g. by executing the CLI with following options:

  ```
  1   bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --add --
  ```

  You can add acls on prefixed resource patterns, e.g. suppose you want to add an acl "Principal User:Jane is allc
  any host". You can do that by executing the CLI with following options:

  ```
  1   bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --add --
  ```

  Note, --resource-pattern-type defaults to 'literal', which only affects resources with the exact same name or, in tl
  name.
- **Removing Acls**
  Removing acls is pretty much the same. The only difference is instead of --add option users will have to specify

above we can execute the CLI with following options:

```
1  bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --remove
```

If you wan to remove the acl added to the prefixed resource pattern above we can execute the CLI with followin

```
1  bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --remove
```

- **List Acls**
  We can list acls for any resource by specifying the --list option with the resource. To list all acls on the literal res options:

```
1  bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --list -
```

  However, this will only return the acls that have been added to this exact resource pattern. Other acls can exist '*', or any acls on prefixed resource patterns. Acls on the wildcard resource pattern can be queried explicitly:

```
1  bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --list -
```

  However, it is not necessarily possible to explicitly query for acls on prefixed resource patterns that match Test- *all* acls affecting Test-topic by using '--resource-pattern-type match', e.g.

```
1  bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --list -
```

  This will list acls on all matching literal, wildcard and prefixed resource patterns.
- **Adding or removing a principal as producer or consumer**
  The most common use case for acl management are adding/removing a principal as producer or consumer so add User:Bob as a producer of Test-topic we can execute the following command:

```
1  bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --add --
```

  Similarly to add Alice as a consumer of Test-topic with consumer group Group-1 we just have to pass --consum

```
1  bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --add --
```

  Note that for consumer option we must also specify the consumer group. In order to remove a principal from pr
- **Admin API based acl management**
  Users having Alter permission on ClusterResource can use Admin API for ACL management. kafka-acls.sh scrip with zookeeper/authorizer directly. All the above examples can be executed by using **--bootstrap-server** option

```
1  bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /tmp/adminclie
2  bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /tmp/adminclie
3  bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /tmp/adminclie
```

Authorization Primitives

Protocol calls are usually performing some operations on certain resources in Kafka. It is required to know the ope we'll list these operations and resources, then list the combination of these with the protocols to see the valid scen

Operations in Kafka

There are a few operation primitives that can be used to build up privileges. These can be matched up with certain are:

- Read
- Write
- Create
- Delete
- Alter
- Describe
- ClusterAction
- DescribeConfigs
- AlterConfigs
- IdempotentWrite
- All

Resources in Kafka

The operations above can be applied on certain resources which are described below.

- **Topic:** this simply represents a Topic. All protocol calls that are acting on topics (such as reading, writing them) authorization error with a topic resource, then a TOPIC_AUTHORIZATION_FAILED (error code: 29) will be returne
- **Group:** this represents the consumer groups in the brokers. All protocol calls that are working with consumer gı subject. If the privilege is not given then a GROUP_AUTHORIZATION_FAILED (error code: 30) will be returned in
- **Cluster:** this resource represents the cluster. Operations that are affecting the whole cluster, like controlled shu is an authorization problem on a cluster resource, then a CLUSTER_AUTHORIZATION_FAILED (error code: 31) w
- **TransactionalId:** this resource represents actions related to transactions, such as committing. If any error occu code: 53) will be returned by brokers.
- **DelegationToken:** this represents the delegation tokens in the cluster. Actions, such as describing delegation tc resource. Since these objects have a little special behavior in Kafka it is recommended to read KIP-48 and the r Tokens.

Operations and Resources on Protocols

In the below table we'll list the valid operations on resources that are executed by the Kafka API protocols.

PROTOCOL (API KEY)	OPERATION	RESOURCE
PRODUCE (0)	Write	TransactionalId
PRODUCE (0)	IdempotentWrite	Cluster
PRODUCE (0)	Write	Topic
FETCH (1)	ClusterAction	Cluster
FETCH (1)	Read	Topic
LIST_OFFSETS (2)	Describe	Topic
METADATA (3)	Describe	Topic

PROTOCOL (API KEY)	OPERATION	RESOURCE
METADATA (3)	Create	Cluster
METADATA (3)	Create	Topic
LEADER_AND_ISR (4)	ClusterAction	Cluster
STOP_REPLICA (5)	ClusterAction	Cluster
UPDATE_METADATA (6)	ClusterAction	Cluster
CONTROLLED_SHUTDOWN (7)	ClusterAction	Cluster
OFFSET_COMMIT (8)	Read	Group
OFFSET_COMMIT (8)	Read	Topic
OFFSET_FETCH (9)	Describe	Group
OFFSET_FETCH (9)	Describe	Topic
FIND_COORDINATOR (10)	Describe	Group
FIND_COORDINATOR (10)	Describe	TransactionalId
JOIN_GROUP (11)	Read	Group
HEARTBEAT (12)	Read	Group
LEAVE_GROUP (13)	Read	Group
SYNC_GROUP (14)	Read	Group
DESCRIBE_GROUPS (15)	Describe	Group
LIST_GROUPS (16)	Describe	Cluster
LIST_GROUPS (16)	Describe	Group
SASL_HANDSHAKE (17)		
API_VERSIONS (18)		

PROTOCOL (API KEY)	OPERATION	RESOURCE
CREATE_TOPICS (19)	Create	Cluster
CREATE_TOPICS (19)	Create	Topic
DELETE_TOPICS (20)	Delete	Topic
DELETE_RECORDS (21)	Delete	Topic
INIT_PRODUCER_ID (22)	Write	TransactionalId
INIT_PRODUCER_ID (22)	IdempotentWrite	Cluster
OFFSET_FOR_LEADER_EPOCH (23)	ClusterAction	Cluster
OFFSET_FOR_LEADER_EPOCH (23)	Describe	Topic
ADD_PARTITIONS_TO_TXN (24)	Write	TransactionalId
ADD_PARTITIONS_TO_TXN (24)	Write	Topic
ADD_OFFSETS_TO_TXN (25)	Write	TransactionalId
ADD_OFFSETS_TO_TXN (25)	Read	Group
END_TXN (26)	Write	TransactionalId
WRITE_TXN_MARKERS (27)	ClusterAction	Cluster
TXN_OFFSET_COMMIT (28)	Write	TransactionalId
TXN_OFFSET_COMMIT (28)	Read	Group
TXN_OFFSET_COMMIT (28)	Read	Topic
DESCRIBE_ACLS (29)	Describe	Cluster
CREATE_ACLS (30)	Alter	Cluster
DELETE_ACLS (31)	Alter	Cluster
DESCRIBE_CONFIGS (32)	DescribeConfigs	Cluster
DESCRIBE_CONFIGS (32)	DescribeConfigs	Topic
ALTER_CONFIGS (33)	AlterConfigs	Cluster
ALTER_CONFIGS (33)	AlterConfigs	Topic
ALTER_REPLICA_LOG_DIRS (34)	Alter	Cluster
DESCRIBE_LOG_DIRS (35)	Describe	Cluster
SASL_AUTHENTICATE (36)		
CREATE_PARTITIONS (37)	Alter	Topic
CREATE_DELEGATION_TOKEN (38)		
RENEW_DELEGATION_TOKEN (39)		

PROTOCOL (API KEY)	OPERATION	RESOURCE
EXPIRE_DELEGATION_TOKEN (40)		
DESCRIBE_DELEGATION_TOKEN (41)	Describe	DelegationToken
DELETE_GROUPS (42)	Delete	Group
ELECT_PREFERRED_LEADERS (43)	ClusterAction	Cluster
INCREMENTAL_ALTER_CONFIGS (44)	AlterConfigs	Cluster
INCREMENTAL_ALTER_CONFIGS (44)	AlterConfigs	Topic
ALTER_PARTITION_REASSIGNMENTS (45)	Alter	Cluster
LIST_PARTITION_REASSIGNMENTS (46)	Describe	Cluster
OFFSET_DELETE (47)	Delete	Group
OFFSET_DELETE (47)	Read	Topic

7.5 Incorporating Security Features in a Running Cluster

You can secure a running cluster via one or more of the supported protocols discussed previously. This is done in

- Incrementally bounce the cluster nodes to open additional secured port(s).
- Restart clients using the secured rather than PLAINTEXT port (assuming you are securing the client-broker con
- Incrementally bounce the cluster again to enable broker-to-broker security (if this is required)
- A final incremental bounce to close the PLAINTEXT port.

The specific steps for configuring SSL and SASL are described in sections 7.2 and 7.3. Follow these steps to enab

The security implementation lets you configure different protocols for both broker-client and broker-broker commu PLAINTEXT port must be left open throughout so brokers and/or clients can continue to communicate.

When performing an incremental bounce stop the brokers cleanly via a SIGTERM. It's also good practice to wait fo next node.

As an example, say we wish to encrypt both broker-client and broker-broker communication with SSL. In the first ir

```
listeners=PLAINTEXT://broker1:9091,SSL://broker1:9092
```

We then restart the clients, changing their config to point at the newly opened, secured port:

```
bootstrap.servers = [broker1:9092,...]
security.protocol = SSL
...etc
```

In the second incremental server bounce we instruct Kafka to use SSL as the broker-broker protocol (which will us

```
listeners=PLAINTEXT://broker1:9091,SSL://broker1:9092
security.inter.broker.protocol=SSL
```

In the final bounce we secure the cluster by closing the PLAINTEXT port:

```
listeners=SSL://broker1:9092
security.inter.broker.protocol=SSL
```

Alternatively we might choose to open multiple ports so that different protocols can be used for broker-broker and encryption throughout (i.e. for broker-broker and broker-client communication) but we'd like to add SASL authentic by opening two additional ports during the first bounce:

```
listeners=PLAINTEXT://broker1:9091,SSL://broker1:9092,SASL_SSL://brok
```

We would then restart the clients, changing their config to point at the newly opened, SASL & SSL secured port:

```
bootstrap.servers = [broker1:9093,...]
security.protocol = SASL_SSL
...etc
```

The second server bounce would switch the cluster to use encrypted broker-broker communication via the SSL po

```
listeners=PLAINTEXT://broker1:9091,SSL://broker1:9092,SASL_SSL://brok
security.inter.broker.protocol=SSL
```

The final bounce secures the cluster by closing the PLAINTEXT port.

```
listeners=SSL://broker1:9092,SASL_SSL://broker1:9093
security.inter.broker.protocol=SSL
```

ZooKeeper can be secured independently of the Kafka cluster. The steps for doing this are covered in section 7.6.:

7.6 ZooKeeper Authentication

7.6.1 New clusters

To enable ZooKeeper authentication on brokers, there are two necessary steps:

1. Create a JAAS login file and set the appropriate system property to point to it as described above
2. Set the configuration property `zookeeper.set.acl` in each broker to true

The metadata stored in ZooKeeper for the Kafka cluster is world-readable, but can only be modified by the brokers
ZooKeeper is not sensitive, but inappropriate manipulation of that data can cause cluster disruption. We also reco
segmentation (only brokers and some admin tools need access to ZooKeeper).

7.6.2 Migrating clusters

If you are running a version of Kafka that does not support security or simply with security disabled, and you want
steps to enable ZooKeeper authentication with minimal disruption to your operations:

1. Perform a rolling restart setting the JAAS login file, which enables brokers to authenticate. At the end of the r
   ACLs, but they will not create znodes with those ACLs
2. Perform a second rolling restart of brokers, this time setting the configuration parameter `zookeeper.set.acl` t
   znodes
3. Execute the ZkSecurityMigrator tool. To execute the tool, there is this script: `./bin/zookeeper-security-migra`
   corresponding sub-trees changing the ACLs of the znodes

It is also possible to turn off authentication in a secure cluster. To do it, follow these steps:

1. Perform a rolling restart of brokers setting the JAAS login file, which enables brokers to authenticate, but sett
   brokers stop creating znodes with secure ACLs, but are still able to authenticate and manipulate all znodes
2. Execute the ZkSecurityMigrator tool. To execute the tool, run this script `./bin/zookeeper-security-migration`
   corresponding sub-trees changing the ACLs of the znodes
3. Perform a second rolling restart of brokers, this time omitting the system property that sets the JAAS login fil

Here is an example of how to run the migration tool:

```
./bin/zookeeper-security-migration.sh --zookeeper.acl=secure --zookeeper.connect=loca
```

Run this to see the full list of parameters:

```
./bin/zookeeper-security-migration.sh --help
```

7.6.3 Migrating the ZooKeeper ensemble

It is also necessary to enable authentication on the ZooKeeper ensemble. To do it, we need to perform a rolling res
ZooKeeper documentation for more detail:

1. [Apache ZooKeeper documentation]

2. [Apache ZooKeeper wiki]

8. KAFKA CONNECT

8.1 Overview

Kafka Connect is a tool for scalably and reliably streaming data between Apache Kafka and other systems. It mak collections of data into and out of Kafka. Kafka Connect can ingest entire databases or collect metrics from all yo for stream processing with low latency. An export job can deliver data from Kafka topics into secondary storage a

Kafka Connect features include:

- **A common framework for Kafka connectors** - Kafka Connect standardizes integration of other data systems w management
- **Distributed and standalone modes** - scale up to a large, centrally managed service supporting an entire organiz deployments
- **REST interface** - submit and manage connectors to your Kafka Connect cluster via an easy to use REST API
- **Automatic offset management** - with just a little information from connectors, Kafka Connect can manage the not need to worry about this error prone part of connector development
- **Distributed and scalable by default** - Kafka Connect builds on the existing group management protocol. More v
- **Streaming/batch integration** - leveraging Kafka's existing capabilities, Kafka Connect is an ideal solution for bri

8.2 User Guide

The quickstart provides a brief example of how to run a standalone version of Kafka Connect. This section descri detail.

Running Kafka Connect

Kafka Connect currently supports two modes of execution: standalone (single process) and distributed.

In standalone mode all work is performed in a single process. This configuration is simpler to setup and get starte makes sense (e.g. collecting log files), but it does not benefit from some of the features of Kafka Connect such as following command:

```
> bin/connect-standalone.sh config/connect-standalone.properties connector1.propertie
```

The first parameter is the configuration for the worker. This includes settings such as the Kafka connection param The provided example should work well with a local cluster running with the default configuration provided by `co` a different configuration or production deployment. All workers (both standalone and distributed) require a few co

- `bootstrap.servers` - List of Kafka servers used to bootstrap connections to Kafka
- `key.converter` - Converter class used to convert between Kafka Connect format and the serialized form t messages written to or read from Kafka, and since this is independent of connectors it allows any connector to include JSON and Avro.

- `value.converter` - Converter class used to convert between Kafka Connect format and the serialized forr messages written to or read from Kafka, and since this is independent of connectors it allows any connector to include JSON and Avro.

The important configuration options specific to standalone mode are:

- `offset.storage.file.filename` - File to store offset data in

The parameters that are configured here are intended for producers and consumers used by Kafka Connect to acc Kafka source and Kafka sink tasks, the same parameters can be used but need to be prefixed with `consumer.` inherited from the worker configuration is `bootstrap.servers` , which in most cases will be sufficient, since tl is a secured cluster, which requires extra parameters to allow connections. These parameters will need to be set u access, once for Kafka sinks and once for Kafka sources.

The remaining parameters are connector configuration files. You may include as many as you want, but all will exe

Distributed mode handles automatic balancing of work, allows you to scale up (or down) dynamically, and offers fa offset commit data. Execution is very similar to standalone mode:

```
> bin/connect-distributed.sh config/connect-distributed.properties
```

The difference is in the class which is started and the configuration parameters which change how the Kafka Conr work, and where to store offsets and task statues. In the distributed mode, Kafka Connect stores the offsets, confi manually create the topics for offset, configs and statuses in order to achieve the desired the number of partitions starting Kafka Connect, the topics will be auto created with default number of partitions and replication factor, whi

In particular, the following configuration parameters, in addition to the common settings mentioned above, are crit

- `group.id` (default `connect-cluster`) - unique name for the cluster, used in forming the Connect cluste
- `config.storage.topic` (default `connect-configs`) - topic to use for storing connector and task con replicated, compacted topic. You may need to manually create the topic to ensure the correct configuration as a automatically configured for deletion rather than compaction
- `offset.storage.topic` (default `connect-offsets`) - topic to use for storing offsets; this topic shoul compaction
- `status.storage.topic` (default `connect-status`) - topic to use for storing statuses; this topic can h compaction

Note that in distributed mode the connector configurations are not passed on the command line. Instead, use the connectors.

Configuring Connectors

Connector configurations are simple key-value mappings. For standalone mode these are defined in a properties fi distributed mode, they will be included in the JSON payload for the request that creates (or modifies) the connecto

Most configurations are connector dependent, so they can't be outlined here. However, there are a few common op

- `name` - Unique name for the connector. Attempting to register again with the same name will fail.

- `connector.class` - The Java class for the connector
- `tasks.max` - The maximum number of tasks that should be created for this connector. The connector may c
- `key.converter` - (optional) Override the default key converter set by the worker.
- `value.converter` - (optional) Override the default value converter set by the worker.

The `connector.class` config supports several formats: the full name or alias of the class for this connector. I org.apache.kafka.connect.file.FileStreamSinkConnector, you can either specify this full name or use FileStreamSin shorter.

Sink connectors also have a few additional options to control their input. Each sink connector must set one of the

- `topics` - A comma-separated list of topics to use as input for this connector
- `topics.regex` - A Java regular expression of topics to use as input for this connector

For any other options, you should consult the documentation for the connector.

Transformations

Connectors can be configured with transformations to make lightweight message-at-a-time modifications. They ca

A transformation chain can be specified in the connector configuration.

- `transforms` - List of aliases for the transformation, specifying the order in which the transformations will be
- `transforms.$alias.type` - Fully qualified class name for the transformation.
- `transforms.$alias.$transformationSpecificConfig` Configuration properties for the transformat

For example, lets take the built-in file source connector and use a transformation to add a static field.

Throughout the example we'll use schemaless JSON data format. To use schemaless format, we changed the foll true to false:

```
key.converter.schemas.enable
value.converter.schemas.enable
```

The file source connector reads each line as a String. We will wrap each line in a Map and then add a second field transformations:

- **HoistField** to place the input line inside a Map
- **InsertField** to add the static field. In this example we'll indicate that the record came from a file connector

After adding the transformations, `connect-file-source.properties` file looks as following:

```
name=local-file-source
connector.class=FileStreamSource
tasks.max=1
file=test.txt
topic=connect-test
transforms=MakeMap, InsertSource
transforms.MakeMap.type=org.apache.kafka.connect.transforms.HoistField$Value
transforms.MakeMap.field=line
```

```
transforms.InsertSource.type=org.apache.kafka.connect.transforms.InsertField$Value
transforms.InsertSource.static.field=data_source
transforms.InsertSource.static.value=test-file-source
```

All the lines starting with `transforms` were added for the transformations. You can see the two transformatior chose to give the transformations. The transformation types are based on the list of built-in transformations you c configuration: HoistField requires a configuration called "field", which is the name of the field in the map that will ir lets us specify the field name and the value that we are adding.

When we ran the file source connector on my sample file without the transformations, and then read them using `I`

```
"foo"
"bar"
"hello world"
```

We then create a new file connector, this time after adding the transformations to the configuration file. This time,

```
{"line":"foo","data_source":"test-file-source"}
{"line":"bar","data_source":"test-file-source"}
{"line":"hello world","data_source":"test-file-source"}
```

You can see that the lines we've read are now part of a JSON map, and there is an extra field with the static value v transformations.

Several widely-applicable data and routing transformations are included with Kafka Connect:

- InsertField - Add a field using either static data or record metadata
- ReplaceField - Filter or rename fields
- MaskField - Replace field with valid null value for the type (0, empty string, etc)
- ValueToKey
- HoistField - Wrap the entire event as a single field inside a Struct or a Map
- ExtractField - Extract a specific field from Struct and Map and include only this field in results
- SetSchemaMetadata - modify the schema name or version
- TimestampRouter - Modify the topic of a record based on original topic and timestamp. Useful when using a si timestamps
- RegexRouter - modify the topic of a record based on original topic, replacement string and a regular expression

Details on how to configure each transformation are listed below:

org.apache.kafka.connect.transforms.InsertField

Insert field(s) using attributes from the record metadata or a configured static value.

Use the concrete transformation type designed for the record key (`org.apache.kafka.connect.transform` (`org.apache.kafka.connect.transforms.InsertField$Value`).

offset.field: Field name for Kafka offset - only applicable to sink connectors.
Suffix with `!` to make this a required field, or `?` to keep it optional (the default).

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

partition.field: Field name for Kafka partition. Suffix with `!` to make this a required field, or `?` to keep it o

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

static.field: Field name for static data field. Suffix with `!` to make this a required field, or `?` to keep it opti

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

static.value: Static field value, if field name configured.

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

timestamp.field: Field name for record timestamp. Suffix with `!` to make this a required field, or `?` to kee

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

topic.field: Field name for Kafka topic. Suffix with `!` to make this a required field, or `?` to keep it optional

Type: string — **Default**: null — **Valid Values**: — **Importance**: medium

---

org.apache.kafka.connect.transforms.ReplaceField

Filter or rename fields.

Use the concrete transformation type designed for the record key (`org.apache.kafka.connect.transform`
(`org.apache.kafka.connect.transforms.ReplaceField$Value`).

blacklist: Fields to exclude. This takes precedence over the whitelist.

Type: list — **Default**: "" — **Valid Values**: — **Importance**: medium

---

renames: Field rename mappings.

Type: list — **Default**: "" — **Valid Values**: list of colon-delimited pairs, e.g. `foo:bar,abc:xyz` — **Impo**

---

whitelist: Fields to include. If specified, only these fields will be used.

Type: list — **Default**: "" — **Valid Values**: — **Importance**: medium

---

org.apache.kafka.connect.transforms.MaskField

Mask specified fields with a valid null value for the field type (i.e. 0, false, empty string, and so on).

Use the concrete transformation type designed for the record key (`org.apache.kafka.connect.transform`
(`org.apache.kafka.connect.transforms.MaskField$Value`).

fields: Names of fields to mask.

Type: list — **Default**: — **Valid Values**: non-empty list — **Importance**: high

---

org.apache.kafka.connect.transforms.ValueToKey

Replace the record key with a new key formed from a subset of fields in the record value.

fields: Field names on the record value to extract as the record key.

Type: list — **Default**: — **Valid Values**: non-empty list — **Importance**: high

---

org.apache.kafka.connect.transforms.HoistField

Wrap data using the specified field name in a Struct when schema present, or a Map in the case of schemaless da

Use the concrete transformation type designed for the record key (`org.apache.kafka.connect.transform`
(`org.apache.kafka.connect.transforms.HoistField$Value`).

field: Field name for the single field that will be created in the resulting Struct or Map.

Type: string — **Default**: — **Valid Values**: — **Importance**: medium

---

org.apache.kafka.connect.transforms.ExtractField

Extract the specified field from a Struct when schema present, or a Map in the case of schemaless data. Any null v

Use the concrete transformation type designed for the record key (`org.apache.kafka.connect.transform`
(`org.apache.kafka.connect.transforms.ExtractField$Value`).

field: Field name to extract.

Type: string — **Default**: — **Valid Values**: — **Importance**: medium

---

org.apache.kafka.connect.transforms.SetSchemaMetadata

Set the schema name, version or both on the record's key (`org.apache.kafka.connect.transforms.SetS`
(`org.apache.kafka.connect.transforms.SetSchemaMetadata$Value`) schema.

schema.name: Schema name to set.

Type: string — **Default**: null — **Valid Values**: — **Importance**: high

---

schema.version: Schema version to set.

Type: int — **Default**: null — **Valid Values**: — **Importance**: high

---

org.apache.kafka.connect.transforms.TimestampRouter

Update the record's topic field as a function of the original topic value and the record timestamp.

This is mainly useful for sink connectors, since the topic field is often used to determine the equivalent entity nam
name).

timestamp.format: Format string for the timestamp that is compatible with `java.text.SimpleDateFor`

Type: string — **Default**: yyyyMMdd — **Valid Values**: — **Importance**: high

---

topic.format: Format string which can contain `${topic}` and `${timestamp}` as placeholders for the t

Type: string — **Default**: ${topic}-${timestamp} — **Valid Values**: — **Importance**: high

---

org.apache.kafka.connect.transforms.RegexRouter

Update the record topic using the configured regular expression and replacement string.

Under the hood, the regex is compiled to a `java.util.regex.Pattern`. If the pattern matches the input topi
with the replacement string to obtain the new topic.

regex: Regular expression to use for matching.

Type: string — **Default**: — **Valid Values**: valid regex — **Importance**: high

---

replacement: Replacement string.

Type: string — **Default**: — **Valid Values**: — **Importance**: high

---

org.apache.kafka.connect.transforms.Flatten

Flatten a nested data structure, generating names for each field by concatenating the field names at each level wit
schema present, or a Map in the case of schemaless data. The default delimiter is '.'.

Use the concrete transformation type designed for the record key (`org.apache.kafka.connect.transform`
(`org.apache.kafka.connect.transforms.Flatten$Value`).

delimiter: Delimiter to insert between field names from the input record when generating field names for the

Type: string — **Default**: . — **Valid Values**: — **Importance**: medium

---

org.apache.kafka.connect.transforms.Cast

Cast fields or the entire key or value to a specific type, e.g. to force an integer field to a smaller width. Only simple

Use the concrete transformation type designed for the record key (`org.apache.kafka.connect.transform`
(`org.apache.kafka.connect.transforms.Cast$Value`).

spec: List of fields and the type to cast them to of the form field1:type,field2:type to cast fields of Maps or St
int16, int32, int64, float32, float64, boolean, and string.

Type: list — **Default**: — **Valid Values**: list of colon-delimited pairs, e.g. `foo:bar,abc:xyz` — **Importa**

---

org.apache.kafka.connect.transforms.TimestampConverter

Convert timestamps between different formats such as Unix epoch, strings, and Connect Date/Timestamp types.

Use the concrete transformation type designed for the record key (`org.apache.kafka.connect.transform`
(`org.apache.kafka.connect.transforms.TimestampConverter$Value`).

target.type: The desired timestamp representation: string, unix, Date, Time, or Timestamp

Type: string — **Default**: — **Valid Values**: — **Importance**: high

---

field: The field containing the timestamp, or empty if the entire value is a timestamp

Type: string — **Default**: "" — **Valid Values**: — **Importance**: high

---

format: A SimpleDateFormat-compatible format for the timestamp. Used to generate the output when type=s

Type: string — **Default**: "" — **Valid Values**: — **Importance**: medium

---

REST API

Since Kafka Connect is intended to be run as a service, it also provides a REST API for managing connectors. The
configuration option. This field should contain a list of listeners in the following format: `protocol://host:po`
are `http` and `https` . For example:

```
listeners=http://localhost:8080,https://localhost:8443
```

By default, if no `listeners` are specified, the REST server runs on port 8083 using the HTTP protocol. When us By default, it will use the `ssl.*` settings. In case it is needed to use different configuration for the REST API tha `listeners.https`. When using the prefix, only the prefixed options will be used and the `ssl.*` options witl configure HTTPS for the REST API:

- `ssl.keystore.location`
- `ssl.keystore.password`
- `ssl.keystore.type`
- `ssl.key.password`
- `ssl.truststore.location`
- `ssl.truststore.password`
- `ssl.truststore.type`
- `ssl.enabled.protocols`
- `ssl.provider`
- `ssl.protocol`
- `ssl.cipher.suites`
- `ssl.keymanager.algorithm`
- `ssl.secure.random.implementation`
- `ssl.trustmanager.algorithm`
- `ssl.endpoint.identification.algorithm`
- `ssl.client.auth`

The REST API is used not only by users to monitor / manage Kafka Connect. It is also used for the Kafka Connect nodes REST API will be forwarded to the leader node REST API. In case the URI under which is given host reachabl options `rest.advertised.host.name`, `rest.advertised.port` and `rest.advertised.listener` nodes to connect with the leader. When using both HTTP and HTTPS listeners, the `rest.advertised.listen` for the cross-cluster communication. When using HTTPS for communication between nodes, the same `ssl.*` (HTTPS client.

The following are the currently supported REST API endpoints:

- `GET /connectors` - return a list of active connectors
- `POST /connectors` - create a new connector; the request body should be a JSON object containing a strin configuration parameters
- `GET /connectors/{name}` - get information about a specific connector
- `GET /connectors/{name}/config` - get the configuration parameters for a specific connector
- `PUT /connectors/{name}/config` - update the configuration parameters for a specific connector
- `GET /connectors/{name}/status` - get current status of the connector, including if it is running, failed, p has failed, and the state of all its tasks
- `GET /connectors/{name}/tasks` - get a list of tasks currently running for a connector
- `GET /connectors/{name}/tasks/{taskid}/status` - get current status of the task, including if it is r information if it has failed
- `PUT /connectors/{name}/pause` - pause the connector and its tasks, which stops message processing

- `PUT /connectors/{name}/resume` - resume a paused connector (or do nothing if the connector is not p
- `POST /connectors/{name}/restart` - restart a connector (typically because it has failed)
- `POST /connectors/{name}/tasks/{taskId}/restart` - restart an individual task (typically because
- `DELETE /connectors/{name}` - delete a connector, halting all tasks and deleting its configuration

Kafka Connect also provides a REST API for getting information about connector plugins:

- `GET /connector-plugins` - return a list of connector plugins installed in the Kafka Connect cluster. Note t the request, which means you may see inconsistent results, especially during a rolling upgrade if you add new c
- `PUT /connector-plugins/{connector-type}/config/validate` - validate the provided configurat per config validation, returns suggested values and error messages during validation.

8.3 Connector Development Guide

This guide describes how developers can write new connectors for Kafka Connect to move data between Kafka ar describes how to create a simple connector.

Core Concepts and APIs

Connectors and Tasks

To copy data between Kafka and another system, users create a `Connector` for the system they want to pull da `SourceConnectors` import data from another system (e.g. `JDBCSourceConnector` would import a relatic (e.g. `HDFSSinkConnector` would export the contents of a Kafka topic to an HDFS file).

`Connectors` do not perform any data copying themselves: their configuration describes the data to be copied, set of `Tasks` that can be distributed to workers. These `Tasks` also come in two corresponding flavors: `Sou`

With an assignment in hand, each `Task` must copy its subset of the data to or from Kafka. In Kafka Connect, it s input and output streams consisting of records with consistent schemas. Sometimes this mapping is obvious: eac parsed line forming a record using the same schema and offsets stored as byte offsets in the file. In other cases it can map each table to a stream, but the offset is less clear. One possible mapping uses a timestamp column to ge queried timestamp can be used as the offset.

Streams and Records

Each stream should be a sequence of key-value records. Both the keys and values can have complex structure -- m data structures can be represented as well. The runtime data format does not assume any particular serialization

In addition to the key and value, records (both those generated by sources and those delivered to sinks) have asso periodically commit the offsets of data that have been processed so that in the event of failures, processing can re reprocessing and duplication of events.

Dynamic Connectors

Not all jobs are static, so `Connector` implementations are also responsible for monitoring the external system the `JDBCSourceConnector` example, the `Connector` might assign a set of tables to each `Task` . When a table to one of the `Tasks` by updating its configuration. When it notices a change that requires reconfiguration (and the framework updates any corresponding `Tasks` .

Developing a Simple Connector

Developing a connector only requires implementing two interfaces, the `Connector` and `Task` . A simple exam package. This connector is meant for use in standalone mode and has implementations of a `SourceConnecto` and a `SinkConnector` / `SinkTask` that writes each record to a file.

The rest of this section will walk through some code to demonstrate the key steps in creating a connector, but dev details are omitted for brevity.

Connector Example

We'll cover the `SourceConnector` as a simple example. `SinkConnector` implementations are very similar. `SourceConnector` and add a couple of fields that will store parsed configuration information (the filename to

```
public class FileStreamSourceConnector extends SourceConnector {
    private String filename;
    private String topic;
```

The easiest method to fill in is `taskClass()` , which defines the class that should be instantiated in worker pro

```
@Override
public Class<? extends Task> taskClass() {
    return FileStreamSourceTask.class;
}
```

We will define the `FileStreamSourceTask` class below. Next, we add some standard lifecycle methods, `sta`

:

```
@Override
public void start(Map<String, String> props) {
    // The complete version includes error handling as well.
    filename = props.get(FILE_CONFIG);
    topic = props.get(TOPIC_CONFIG);
}

@Override
public void stop() {
    // Nothing to do since no background monitoring is required.
}
```

Finally, the real core of the implementation is in `taskConfigs()` . In this case we are only handling a single file, per the `maxTasks` argument, we return a list with only one entry:

```
@Override
public List<Map<String, String>> taskConfigs(int maxTasks) {
```

```
    ArrayList<Map<String, String>> configs = new ArrayList<>();
    // Only one input stream makes sense.
    Map<String, String> config = new HashMap<>();
    if (filename != null)
        config.put(FILE_CONFIG, filename);
    config.put(TOPIC_CONFIG, topic);
    configs.add(config);
    return configs;
}
```

Although not used in the example, `SourceTask` also provides two APIs to commit offsets in the source system source systems which have an acknowledgement mechanism for messages. Overriding these methods allows the system, either in bulk or individually, once they have been written to Kafka. The `commit` API stores the offsets in `poll`. The implementation of this API should block until the commit is complete. The `commitRecord` API sa after it is written to Kafka. As Kafka Connect will record offsets automatically, `SourceTask`s are not required to acknowledge messages in the source system, only one of the APIs is typically required.

Even with multiple tasks, this method implementation is usually pretty simple. It just has to determine the number is pulling data from, and then divvy them up. Because some patterns for splitting work among tasks are so comm these cases.

Note that this simple example does not include dynamic input. See the discussion in the next section for how to tr

Task Example - Source Task

Next we'll describe the implementation of the corresponding `SourceTask`. The implementation is short, but too describe most of the implementation, but you can refer to the source code for the full example.

Just as with the connector, we need to create a class inheriting from the appropriate base `Task` class. It also ha

```
public class FileStreamSourceTask extends SourceTask {
    String filename;
    InputStream stream;
    String topic;

    @Override
    public void start(Map<String, String> props) {
        filename = props.get(FileStreamSourceConnector.FILE_CONFIG);
        stream = openOrThrowError(filename);
        topic = props.get(FileStreamSourceConnector.TOPIC_CONFIG);
    }

    @Override
    public synchronized void stop() {
        stream.close();
    }
```

These are slightly simplified versions, but show that these methods should be relatively simple and the only work t two points to note about this implementation. First, the `start()` method does not yet handle resuming from a the `stop()` method is synchronized. This will be necessary because `SourceTasks` are given a dedicated thr with a call from a different thread in the Worker.

Next, we implement the main functionality of the task, the `poll()` method which gets events from the input sys

```
@Override
public List<SourceRecord> poll() throws InterruptedException {
    try {
        ArrayList<SourceRecord> records = new ArrayList<>();
        while (streamValid(stream) && records.isEmpty()) {
            LineAndOffset line = readToNextLine(stream);
            if (line != null) {
                Map<String, Object> sourcePartition = Collections.singletonMap("file
                Map<String, Object> sourceOffset = Collections.singletonMap("positio
                records.add(new SourceRecord(sourcePartition, sourceOffset, topic, S
            } else {
                Thread.sleep(1);
            }
        }
        return records;
    } catch (IOException e) {
        // Underlying stream was killed, probably as a result of calling stop. Allow
        // null, and driving thread will handle any shutdown if necessary.
    }
    return null;
}
```

Again, we've omitted some details, but we can see the important steps: the `poll()` method is going to be calle from the file. For each line it reads, it also tracks the file offset. It uses this information to create an output `Sourc` (there is only one, the single file being read), source offset (byte offset in the file), output topic name, and output va always be a string). Other variants of the `SourceRecord` constructor can also include a specific output partitio

Note that this implementation uses the normal Java `InputStream` interface and may sleep if data is not availa with a dedicated thread. While task implementations have to conform to the basic `poll()` interface, they have based implementation would be more efficient, but this simple approach works, is quick to implement, and is com

Sink Tasks

The previous section described how to implement a simple `SourceTask`. Unlike `SourceConnector` and `S` different interfaces because `SourceTask` uses a pull interface and `SinkTask` uses a push interface. Both sl is quite different:

```
public abstract class SinkTask implements Task {
    public void initialize(SinkTaskContext context) {
        this.context = context;
    }

    public abstract void put(Collection<SinkRecord> records);

    public void flush(Map<TopicPartition, OffsetAndMetadata> currentOffsets) {
    }
```

The `SinkTask` documentation contains full details, but this interface is nearly as simple as the `SourceTask` implementation, accepting sets of `SinkRecords`, performing any required translation, and storing them in the d has been fully written to the destination system before returning. In fact, in many cases internal buffering will be us the overhead of inserting events into the downstream data store. The `SinkRecords` contain essentially the sar offset, the event key and value, and optional headers.

The `flush()` method is used during the offset commit process, which allows tasks to recover from failures and method should push any outstanding data to the destination system and then block until the write has been ackno useful in some cases where implementations want to store offset information in the destination store to provide e and use atomic move operations to make sure the `flush()` operation atomically commits the data and offsets

Resuming from Previous Offsets

The `SourceTask` implementation included a stream ID (the input filename) and offset (position in the file) with periodically so that in the case of a failure, the task can recover and minimize the number of events that are reproc offset if Kafka Connect was stopped gracefully, e.g. in standalone mode or due to a job reconfiguration). This com the connector knows how to seek back to the right position in the input stream to resume from that location.

To correctly resume upon startup, the task can use the `SourceContext` passed into its `initialize()` met a bit more code to read the offset (if it exists) and seek to that position:

```
stream = new FileInputStream(filename);
Map<String, Object> offset = context.offsetStorageReader().offset(Collections.singlet
if (offset != null) {
    Long lastRecordedOffset = (Long) offset.get("position");
    if (lastRecordedOffset != null)
        seekToOffset(stream, lastRecordedOffset);
}
```

Of course, you might need to read many keys for each of the input streams. The `OffsetStorageReader` inter offsets, then apply them by seeking each input stream to the appropriate position.

Dynamic Input/Output Streams

Kafka Connect is intended to define bulk data copying jobs, such as copying an entire database rather than creatir this design is that the set of input or output streams for a connector can vary over time.

Source connectors need to monitor the source system for changes, e.g. table additions/deletions in a database. W `ConnectorContext` object that reconfiguration is necessary. For example, in a `SourceConnector`:

```
if (inputsChanged())
    this.context.requestTaskReconfiguration();
```

The framework will promptly request new configuration information and update the tasks, allowing them to gracef the `SourceConnector` this monitoring is currently left up to the connector implementation. If an extra thread is it itself.

Ideally this code for monitoring changes would be isolated to the `Connector` and tasks would not need to worr commonly when one of their input streams is destroyed in the input system, e.g. if a table is dropped from a datab which will be common if the `Connector` needs to poll for changes, the `Task` will need to handle the subsequ catching and handling the appropriate exception.

`SinkConnectors` usually only have to handle the addition of streams, which may translate to new entries in th any changes to the Kafka input, such as when the set of input topics changes because of a regex subscription. `S creating new resources in the downstream system, such as a new table in a database. The trickiest situation to ha `SinkTasks` seeing a new input stream for the first time and simultaneously trying to create the new resource. special code for handling a dynamic set of streams.

Connect Configuration Validation

Kafka Connect allows you to validate connector configurations before submitting a connector to be executed and take advantage of this, connector developers need to provide an implementation of `config()` to expose the co

The following code in `FileStreamSourceConnector` defines the configuration and exposes it to the framew

```
private static final ConfigDef CONFIG_DEF = new ConfigDef()
    .define(FILE_CONFIG, Type.STRING, Importance.HIGH, "Source filename.")
    .define(TOPIC_CONFIG, Type.STRING, Importance.HIGH, "The topic to publish data to

public ConfigDef config() {
    return CONFIG_DEF;
}
```

`ConfigDef` class is used for specifying the set of expected configurations. For each configuration, you can spe group information, the order in the group, the width of the configuration value and the name suitable for display in configuration validation by overriding the `Validator` class. Moreover, as there may be dependencies between configuration may change according to the values of other configurations. To handle this, `ConfigDef` allows yc implementation of `Recommender` to get valid values and set visibility of a configuration given the current config

Also, the `validate()` method in `Connector` provides a default validation implementation which returns a li recommended values for each configuration. However, it does not use the recommended values for configuration implementation for customized configuration validation, which may use the recommended values.

Working with Schemas

The FileStream connectors are good examples because they are simple, but they also have trivially structured data need schemas with more complex data formats.

To create more complex data, you'll need to work with the Kafka Connect `data` API. Most structured records wil `Schema` and `Struct`.

The API documentation provides a complete reference, but here is a simple example creating a `Schema` and `S

```
Schema schema = SchemaBuilder.struct().name(NAME)
    .field("name", Schema.STRING_SCHEMA)
```

```
    .field("age", Schema.INT_SCHEMA)
    .field("admin", new SchemaBuilder.boolean().defaultValue(false).build())
    .build();

Struct struct = new Struct(schema)
    .put("name", "Barbara Liskov")
    .put("age", 75);
```

If you are implementing a source connector, you'll need to decide when and how to create schemas. Where possib example, if your connector is guaranteed to have a fixed schema, create it statically and reuse a single instance.

However, many connectors will have dynamic schemas. One simple example of this is a database connector. Cons for the entire connector (as it varies from table to table). But it also may not be fixed for a single table over the lifet `TABLE` command. The connector must be able to detect these changes and react appropriately.

Sink connectors are usually simpler because they are consuming data and therefore do not need to create schema schemas they receive have the expected format. When the schema does not match -- usually indicating the upstre translated to the destination system -- sink connectors should throw an exception to indicate this error to the syste

Kafka Connect Administration

Kafka Connect's [REST layer](#) provides a set of APIs to enable administration of the cluster. This includes APIs to vie well as to alter their current behavior (e.g. changing configuration and restarting tasks).

When a connector is first submitted to the cluster, a rebalance is triggered between the Connect workers in order t connector. This same rebalancing procedure is also used when connectors increase or decrease the number of ta when a worker is added or removed from the group as part of an intentional upgrade of the Connect cluster or due

In versions prior to 2.3.0, the Connect workers would rebalance the full set of connectors and their tasks in the clu approximately the same amount of work. This behavior can be still enabled by setting `connect.protocol=eag`

Starting with 2.3.0, Kafka Connect is using by default a protocol that performs [incremental cooperative rebalancin](#) Connect workers, affecting only tasks that are new, to be removed, or need to move from one worker to another. O they would have been with the old protocol.

If a Connect worker leaves the group, intentionally or due to a failure, Connect waits for `scheduled.rebalance` defaults to five minutes (`300000ms`) to tolerate failures or upgrades of workers without immediately redistributi configured delay, it gets its previously assigned tasks in full. However, this means that the tasks will remain unassi `scheduled.rebalance.max.delay.ms` elapses. If a worker does not return within that time limit, Connect v Connect cluster.

The new Connect protocol is enabled when all the workers that form the Connect cluster are configured with `con` when this property is missing. Therefore, upgrading to the new Connect protocol happens automatically when all t cluster will activate incremental cooperative rebalancing when the last worker joins on version 2.3.0.

You can use the REST API to view the current status of a connector and its tasks, including the ID of the worker to `/connectors/file-source/status` request shows the status of a connector named `file-source` :

```
{
"name": "file-source",
"connector": {
    "state": "RUNNING",
    "worker_id": "192.168.1.208:8083"
},
"tasks": [
    {
    "id": 0,
    "state": "RUNNING",
    "worker_id": "192.168.1.209:8083"
    }
]
}
```

Connectors and their tasks publish status updates to a shared topic (configured with `status.storage.topic` consume this topic asynchronously, there is typically a (short) delay before a state change is visible through the st of its tasks:

- **UNASSIGNED:** The connector/task has not yet been assigned to a worker.
- **RUNNING:** The connector/task is running.
- **PAUSED:** The connector/task has been administratively paused.
- **FAILED:** The connector/task has failed (usually by raising an exception, which is reported in the status output).
- **DESTROYED:** The connector/task has been administratively removed and will stop appearing in the Connect clu

In most cases, connector and task states will match, though they may be different for short periods of time when c connector is first started, there may be a noticeable delay before the connector and its tasks have all transitioned t Connect does not automatically restart failed tasks. To restart a connector/task manually, you can use the restart rebalance is taking place, Connect will return a 409 (Conflict) status code. You can retry after the rebalance compl restart all the connectors and tasks in the cluster.

It's sometimes useful to temporarily stop the message processing of a connector. For example, if the remote syste connectors to stop polling it for new data instead of filling logs with exception spam. For this use case, Connect of Connect will stop polling it for additional records. While a sink connector is paused, Connect will stop pushing new restart the cluster, the connector will not begin message processing again until the task has been resumed. Note t transitioned to the PAUSED state since it may take time for them to finish whatever processing they were in the mi transition to the PAUSED state until they have been restarted.

9. KAFKA STREAMS

Kafka Streams is a client library for processing and analyzing data stored in Kafka. It builds upon important strean event time and processing time, windowing support, exactly-once processing semantics and simple yet efficient m

Kafka Streams has a **low barrier to entry**: You can quickly write and run a small-scale proof-of-concept on a single application on multiple machines to scale up to high-volume production workloads. Kafka Streams transparently h application by leveraging Kafka's parallelism model.

Learn More about Kafka Streams read this Section.