



## Red Hat AMQ 7.6

### Using AMQ Streams on RHEL

For use with AMQ Streams 1.4 on Red Hat Enterprise Linux



# Red Hat AMQ 7.6 Using AMQ Streams on RHEL

---

For use with AMQ Streams 1.4 on Red Hat Enterprise Linux

## Legal Notice

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

This guide describes how to install, configure, and manage Red Hat AMQ Streams to build a large-scale messaging network.

## Table of Contents

<b>CHAPTER 1. OVERVIEW OF AMQ STREAMS</b> .....	<b>7</b>
1.1. KAFKA CAPABILITIES	8
1.2. KAFKA USE CASES	8
1.3. SUPPORTED CONFIGURATIONS	9
1.4. DOCUMENT CONVENTIONS	9
<b>CHAPTER 2. GETTING STARTED</b> .....	<b>10</b>
2.1. AMQ STREAMS DISTRIBUTION	10
2.2. DOWNLOADING AN AMQ STREAMS ARCHIVE	10
2.3. INSTALLING AMQ STREAMS	10
2.4. DATA STORAGE CONSIDERATIONS	11
2.4.1. Apache Kafka and ZooKeeper storage support	11
2.4.2. File systems	11
2.5. RUNNING A SINGLE NODE AMQ STREAMS CLUSTER	12
2.6. USING THE CLUSTER	13
2.7. STOPPING THE AMQ STREAMS SERVICES	14
2.8. CONFIGURING AMQ STREAMS	14
<b>CHAPTER 3. CONFIGURING ZOOKEEPER</b> .....	<b>16</b>
3.1. BASIC CONFIGURATION	16
3.2. ZOOKEEPER CLUSTER CONFIGURATION	16
3.3. RUNNING MULTI-NODE ZOOKEEPER CLUSTER	18
3.4. AUTHENTICATION	20
3.4.1. Authentication with SASL	20
3.4.2. Enabling Server-to-server authentication using DIGEST-MD5	22
3.4.3. Enabling Client-to-server authentication using DIGEST-MD5	23
3.5. AUTHORIZATION	24
3.6. TLS	24
3.7. ADDITIONAL CONFIGURATION OPTIONS	25
3.8. LOGGING	25
<b>CHAPTER 4. CONFIGURING KAFKA</b> .....	<b>26</b>
4.1. ZOOKEEPER	26
4.2. LISTENERS	26
4.3. COMMIT LOGS	27
4.4. BROKER ID	27
4.5. RUNNING A MULTI-NODE KAFKA CLUSTER	27
4.6. ZOOKEEPER AUTHENTICATION	29
4.6.1. JAAS Configuration	29
4.6.2. Enabling ZooKeeper authentication	29
4.7. AUTHORIZATION	30
4.7.1. Simple ACL authorizer	30
4.7.1.1. ACL rules	30
4.7.1.2. Principals	31
4.7.1.3. Authentication of users	31
4.7.1.4. Super users	31
4.7.1.5. Replica broker authentication	32
4.7.1.6. Supported resources	32
4.7.1.7. Supported operations	32
4.7.1.8. ACL management options	33
4.7.2. Enabling authorization	37
4.7.3. Adding ACL rules	37

4.7.4. Listing ACL rules	38
4.7.5. Removing ACL rules	39
4.8. ZOOKEEPER AUTHORIZATION	40
4.8.1. ACL Configuration	40
4.8.2. Enabling ZooKeeper ACLs for a new Kafka cluster	40
4.8.3. Enabling ZooKeeper ACLs in an existing Kafka cluster	41
4.9. ENCRYPTION AND AUTHENTICATION	41
4.9.1. Listener configuration	42
4.9.2. TLS Encryption	42
4.9.3. Enabling TLS encryption	43
4.9.4. Authentication	44
4.9.4.1. TLS client authentication	44
4.9.4.2. SASL authentication	45
4.9.5. Enabling TLS client authentication	48
4.9.6. Enabling SASL PLAIN authentication	49
4.9.7. Enabling SASL SCRAM authentication	50
4.9.8. Adding SASL SCRAM users	51
4.9.9. Deleting SASL SCRAM users	51
4.10. USING OAUTH 2.0 TOKEN-BASED AUTHENTICATION	52
4.10.1. OAuth 2.0 authentication mechanism	53
4.10.1.1. Configuring OAuth 2.0 with properties or variables	53
4.10.2. OAuth 2.0 Kafka broker configuration	53
4.10.2.1. OAuth 2.0 client configuration on an authorization server	53
4.10.2.2. OAuth 2.0 authentication configuration in the Kafka cluster	54
4.10.2.3. Fast local JWT token validation configuration	56
4.10.2.4. OAuth 2.0 introspection endpoint configuration	57
4.10.3. OAuth 2.0 Kafka client configuration	58
4.10.4. OAuth 2.0 client authentication flow	59
4.10.4.1. Example client authentication flows	59
4.10.5. Configuring OAuth 2.0 authentication	61
4.10.5.1. Configuring Red Hat Single Sign-On as an OAuth 2.0 authorization server	62
4.10.5.2. Configuring OAuth 2.0 support for Kafka brokers	63
4.10.5.3. Configuring Kafka Java clients to use OAuth 2.0	65
4.11. USING OAUTH 2.0 TOKEN-BASED AUTHORIZATION	66
Trying this feature	66
Authorizing access to Kafka brokers	67
4.11.1. OAuth 2.0 authorization mechanism	67
4.11.1.1. Kafka broker custom authorizer	67
4.11.2. Configuring OAuth 2.0 authorization support	67
4.12. LOGGING	69
<b>CHAPTER 5. TOPICS</b> .....	<b>71</b>
5.1. PARTITIONS AND REPLICAS	71
5.2. MESSAGE RETENTION	71
5.3. TOPIC AUTO-CREATION	72
5.4. TOPIC DELETION	72
5.5. TOPIC CONFIGURATION	72
5.6. INTERNAL TOPICS	73
5.7. CREATING A TOPIC	74
5.8. LISTING AND DESCRIBING TOPICS	75
5.9. MODIFYING A TOPIC CONFIGURATION	75
5.10. DELETING A TOPIC	77

<b>CHAPTER 6. SCALING CLUSTERS</b> .....	<b>78</b>
6.1. SCALING KAFKA CLUSTERS	78
6.1.1. Adding brokers to a cluster	78
6.1.2. Removing brokers from the cluster	78
6.2. REASSIGNMENT OF PARTITIONS	78
6.2.1. Reassignment JSON file	79
6.2.2. Generating reassignment JSON files	79
6.2.3. Creating reassignment JSON files manually	80
6.3. REASSIGNMENT THROTTLES	80
6.4. SCALING UP A KAFKA CLUSTER	80
6.5. SCALING DOWN A KAFKA CLUSTER	82
6.6. SCALING UP A ZOOKEEPER CLUSTER	83
6.7. SCALING DOWN A ZOOKEEPER CLUSTER	84
<b>CHAPTER 7. MONITORING YOUR CLUSTER USING JMX</b> .....	<b>86</b>
7.1. JMX CONFIGURATION OPTIONS	86
7.2. DISABLING THE JMX AGENT	86
7.3. CONNECTING TO THE JVM FROM A DIFFERENT MACHINE	86
7.4. MONITORING USING JCONSOLE	87
7.5. IMPORTANT KAFKA BROKER METRICS	87
7.5.1. Kafka server metrics	87
7.5.2. Kafka network metrics	89
7.5.3. Kafka log metrics	91
7.5.4. Kafka controller metrics	92
7.5.5. Yammer metrics	92
7.6. PRODUCER MBEANS	93
7.6.1. MBeans matching kafka.producer:type=producer-metrics,client-id=*	93
7.6.2. MBeans matching kafka.producer:type=producer-metrics,client-id=*,node-id=*	95
7.6.3. MBeans matching kafka.producer:type=producer-topic-metrics,client-id=*,topic=*	96
7.7. CONSUMER MBEANS	96
7.7.1. MBeans matching kafka.consumer:type=consumer-metrics,client-id=*	97
7.7.2. MBeans matching kafka.consumer:type=consumer-metrics,client-id=*,node-id=*	98
7.7.3. MBeans matching kafka.consumer:type=consumer-coordinator-metrics,client-id=*	98
7.7.4. MBeans matching kafka.consumer:type=consumer-fetch-manager-metrics,client-id=*	99
7.7.5. MBeans matching kafka.consumer:type=consumer-fetch-manager-metrics,client-id=*,topic=*	100
7.7.6. MBeans matching kafka.consumer:type=consumer-fetch-manager-metrics,client-id=*,topic=*,partition=*	100
7.8. KAFKA CONNECT MBEANS	101
7.8.1. MBeans matching kafka.connect:type=connect-metrics,client-id=*	101
7.8.2. MBeans matching kafka.connect:type=connect-metrics,client-id=*,node-id=*	102
7.8.3. MBeans matching kafka.connect:type=connect-worker-metrics	103
7.8.4. MBeans matching kafka.connect:type=connect-worker-rebalance-metrics	104
7.8.5. MBeans matching kafka.connect:type=connector-metrics,connector=*	104
7.8.6. MBeans matching kafka.connect:type=connector-task-metrics,connector=*,task=*	104
7.8.7. MBeans matching kafka.connect:type=sink-task-metrics,connector=*,task=*	105
7.8.8. MBeans matching kafka.connect:type=source-task-metrics,connector=*,task=*	106
7.8.9. MBeans matching kafka.connect:type=task-error-metrics,connector=*,task=*	107
7.9. KAFKA STREAMS MBEANS	108
7.9.1. MBeans matching kafka.streams:type=stream-metrics,client-id=*	108
7.9.2. MBeans matching kafka.streams:type=stream-task-metrics,client-id=*,task-id=*	109
7.9.3. MBeans matching kafka.streams:type=stream-processor-node-metrics,client-id=*,task-id=*,processor-node-id=*	110
7.9.4. MBeans matching kafka.streams:type=stream-[store-scope]-metrics,client-id=*,task-id=*,[store-scope]-	

id=*	111
7.9.5. MBeans matching kafka.streams:type=stream-record-cache-metrics,client-id=*,task-id=*,record-cache-id=*	113
<b>CHAPTER 8. KAFKA CONNECT</b>	<b>114</b>
8.1. KAFKA CONNECT IN STANDALONE MODE	114
8.1.1. Configuring Kafka Connect in standalone mode	114
8.1.2. Configuring connectors in Kafka Connect in standalone mode	115
8.1.3. Running Kafka Connect in standalone mode	115
8.2. KAFKA CONNECT IN DISTRIBUTED MODE	116
8.2.1. Configuring Kafka Connect in distributed mode	116
8.2.2. Configuring connectors in distributed Kafka Connect	117
8.2.3. Running distributed Kafka Connect	118
8.2.4. Creating connectors	119
8.2.5. Deleting connectors	120
8.3. CONNECTOR PLUG-INS	120
8.4. ADDING CONNECTOR PLUGINS	121
<b>CHAPTER 9. USING AMQ STREAMS WITH MIRRORMAKER 2.0</b>	<b>122</b>
9.1. MIRRORMAKER 2.0 DATA REPLICATION	122
9.2. CLUSTER CONFIGURATION	123
9.2.1. Bidirectional replication	123
9.2.2. Topic configuration synchronization	124
9.2.3. Data integrity	124
9.2.4. Offset tracking	124
9.2.5. Connectivity checks	125
9.3. ACL RULES SYNCHRONIZATION	125
9.4. SYNCHRONIZING DATA BETWEEN KAFKA CLUSTERS USING MIRRORMAKER 2.0	125
9.5. USING MIRRORMAKER 2.0 IN LEGACY MODE	127
<b>CHAPTER 10. KAFKA CLIENTS</b>	<b>129</b>
10.1. ADDING KAFKA CLIENTS AS A DEPENDENCY TO YOUR MAVEN PROJECT	129
<b>CHAPTER 11. KAFKA STREAMS API OVERVIEW</b>	<b>131</b>
11.1. ADDING THE KAFKA STREAMS API AS A DEPENDENCY TO YOUR MAVEN PROJECT	131
<b>CHAPTER 12. KAFKA BRIDGE</b>	<b>133</b>
12.1. KAFKA BRIDGE OVERVIEW	133
12.1.1. Requests to the Kafka Bridge	133
12.1.1.1. Authentication and encryption	133
12.1.1.2. Data formats and headers	134
12.1.1.2.1. Content Type headers	134
12.1.1.2.2. Embedded data format	134
12.1.1.2.3. Accept headers	135
12.1.2. Configuring loggers for the Kafka Bridge	135
12.1.3. Kafka Bridge API resources	136
12.1.4. Downloading a Kafka Bridge archive	137
12.1.5. Configuring Kafka Bridge properties	137
12.1.6. Installing the Kafka Bridge	138
12.2. KAFKA BRIDGE QUICKSTART	138
12.2.1. Deploying the Kafka Bridge locally	139
12.2.2. Producing messages to topics and partitions	139
12.2.3. Creating a Kafka Bridge consumer	141
12.2.4. Subscribing a Kafka Bridge consumer to topics	142



12.2.5. Retrieving the latest messages from a Kafka Bridge consumer	143
12.2.6. Committing offsets to the log	144
12.2.7. Seeking to offsets for a partition	144
12.2.8. Deleting a Kafka Bridge consumer	145
<b>CHAPTER 13. DISTRIBUTED TRACING</b>	<b>147</b>
13.1. OVERVIEW OF DISTRIBUTED TRACING	147
13.1.1. OpenTracing and Jaeger	147
13.2. CONFIGURING KAFKA CLIENTS FOR TRACING	148
13.2.1. Enabling a Jaeger tracer for Kafka clients	148
13.2.2. Instrumenting Kafka Producers and Consumers for tracing	149
13.2.2.1. Custom span names in a Decorator pattern	151
13.2.2.2. Built-in span names	152
13.2.3. Instrumenting Kafka Streams applications for tracing	152
13.3. CONFIGURING MIRRORMAKER AND KAFKA CONNECT FOR TRACING	153
13.3.1. Enabling tracing for MirrorMaker	153
13.3.2. Enabling tracing for Kafka Connect	153
13.4. ENABLING TRACING FOR THE KAFKA BRIDGE	154
13.5. TRACING ENVIRONMENT VARIABLES	155
<b>CHAPTER 14. KAFKA EXPORTER</b>	<b>157</b>
14.1. CONSUMER LAG	157
14.2. KAFKA EXPORTER ALERTING RULE EXAMPLES	158
14.3. KAFKA EXPORTER METRICS	158
14.4. RUNNING KAFKA EXPORTER	159
14.5. PRESENTING KAFKA EXPORTER METRICS IN GRAFANA	160
<b>CHAPTER 15. AMQ STREAMS AND KAFKA UPGRADES</b>	<b>162</b>
15.1. UPGRADE PREREQUISITES	162
15.2. UPGRADE PROCESS	162
15.3. KAFKA VERSIONS	162
15.4. UPGRADING TO AMQ STREAMS 1.4	163
15.4.1. Upgrading ZooKeeper	163
15.4.2. Upgrading Kafka brokers	165
15.4.3. Upgrading Kafka Connect	167
15.5. UPGRADING KAFKA	168
15.5.1. Upgrading Kafka brokers to use the new inter-broker protocol version	168
15.5.2. Strategies for upgrading clients	169
15.5.3. Upgrading client applications to the new Kafka version	171
15.5.4. Upgrading consumers and Kafka Streams applications to cooperative rebalancing	172
15.5.5. Upgrading Kafka brokers to use the new message format version	173
<b>APPENDIX A. BROKER CONFIGURATION PARAMETERS</b>	<b>175</b>
<b>APPENDIX B. TOPIC CONFIGURATION PARAMETERS</b>	<b>207</b>
<b>APPENDIX C. CONSUMER CONFIGURATION PARAMETERS</b>	<b>213</b>
<b>APPENDIX D. PRODUCER CONFIGURATION PARAMETERS</b>	<b>225</b>
<b>APPENDIX E. ADMIN CLIENT CONFIGURATION PARAMETERS</b>	<b>236</b>
<b>APPENDIX F. KAFKA CONNECT CONFIGURATION PARAMETERS</b>	<b>243</b>
<b>APPENDIX G. KAFKA STREAMS CONFIGURATION PARAMETERS</b>	<b>256</b>



# CHAPTER 1. OVERVIEW OF AMQ STREAMS

Red Hat AMQ Streams is a massively-scalable, distributed, and high-performance data streaming platform based on the Apache ZooKeeper and Apache Kafka projects.

The main components comprise:

## **Kafka Broker**

Messaging broker responsible for delivering records from producing clients to consuming clients. Apache ZooKeeper is a core dependency for Kafka, providing a cluster coordination service for highly reliable distributed coordination.

## **Kafka Streams API**

API for writing *stream processor* applications.

## **Producer and Consumer APIs**

Java-based APIs for producing and consuming messages to and from Kafka brokers.

## **Kafka Bridge**

AMQ Streams Kafka Bridge provides a RESTful interface that allows HTTP-based clients to interact with a Kafka cluster.

## **Kafka Connect**

A toolkit for streaming data between Kafka brokers and other systems using *Connector* plugins.

## **Kafka MirrorMaker**

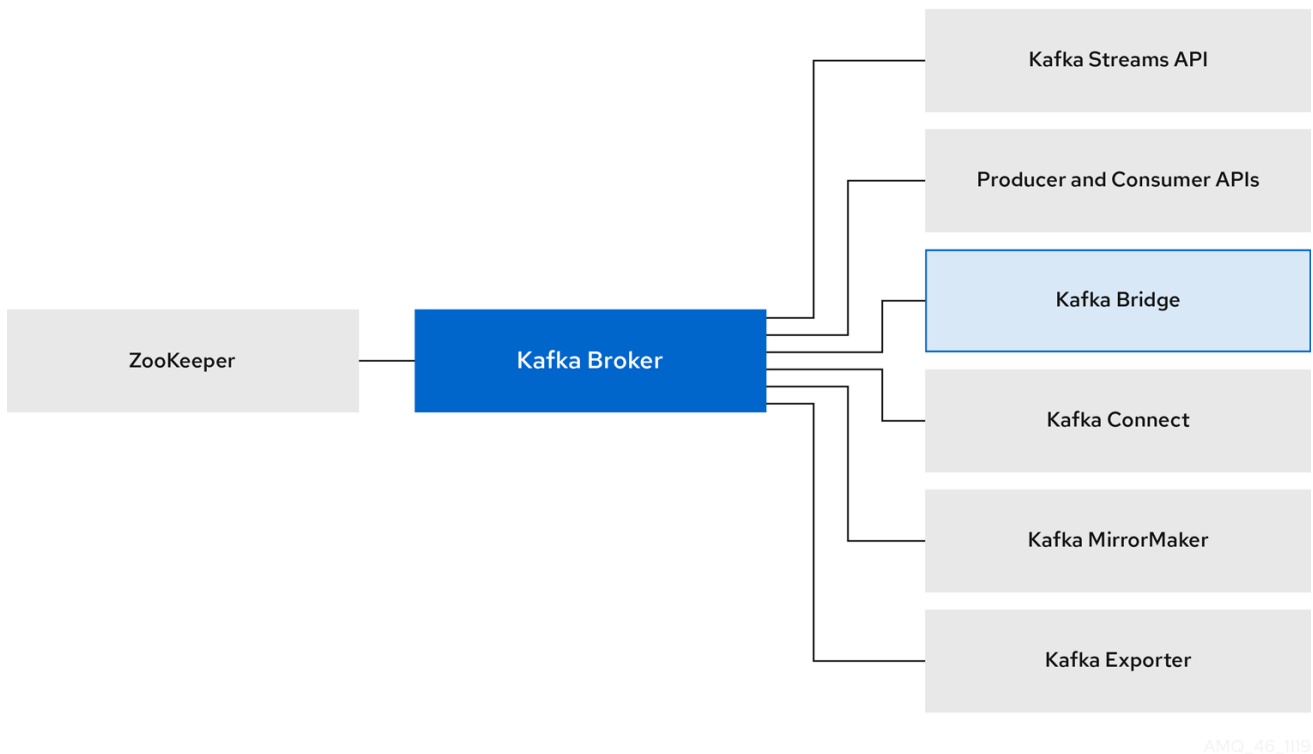
Replicates data between two Kafka clusters, within or across data centers.

## **Kafka Exporter**

An exporter used in the extraction of Kafka metrics data for monitoring.

A cluster of Kafka brokers is the hub connecting all these components. The broker uses Apache ZooKeeper for storing configuration data and for cluster coordination. Before running Apache Kafka, an Apache ZooKeeper cluster has to be ready.

Figure 1.1. AMQ Streams architecture



## 1.1. KAFKA CAPABILITIES

The underlying data stream-processing capabilities and component architecture of Kafka can deliver:

- Microservices and other applications to share data with extremely high throughput and low latency
- Message ordering guarantees
- Message rewind/replay from data storage to reconstruct an application state
- Message compaction to remove old records when using a key-value log
- Horizontal scalability in a cluster configuration
- Replication of data to control fault tolerance
- Retention of high volumes of data for immediate access

## 1.2. KAFKA USE CASES

Kafka's capabilities make it suitable for:

- Event-driven architectures
- Event sourcing to capture changes to the state of an application as a log of events
- Message brokering
- Website activity tracking

- Operational monitoring through metrics
- Log collection and aggregation
- Commit logs for distributed systems
- Stream processing so that applications can respond to data in real time

### 1.3. SUPPORTED CONFIGURATIONS

In order to be running in a supported configuration, AMQ Streams must be running in one of the following JVM versions and on one of the supported operating systems.

**Table 1.1. List of supported Java Virtual Machines**

Java Virtual Machine	Version
OpenJDK	1.8, 11 <sup>[a]</sup>
OracleJDK	1.8
IBM JDK	1.8
<sup>[a]</sup> For RHEL 8.x	

**Table 1.2. List of supported Operating Systems**

Operating System	Architecture	Version
Red Hat Enterprise Linux	x86_64	7.x, 8.x

### 1.4. DOCUMENT CONVENTIONS

#### Replaceables

In this document, replaceable text is styled in monospace and surrounded by angle brackets.

For example, in the following code, you will want to replace **<bootstrap-address>** and **<topic-name>** with your own address and topic name:

```
bin/kafka-console-consumer.sh --bootstrap-server <bootstrap-address> --topic <topic-name> --from-beginning
```

## CHAPTER 2. GETTING STARTED

### 2.1. AMQ STREAMS DISTRIBUTION

AMQ Streams is distributed as single ZIP file. This ZIP file contains AMQ Streams components:

- Apache ZooKeeper
- Apache Kafka
- Apache Kafka Connect
- Apache Kafka MirrorMaker
- [Kafka Bridge](#)
- [Kafka Exporter](#)

### 2.2. DOWNLOADING AN AMQ STREAMS ARCHIVE

An archived distribution of AMQ Streams is available for download from the Red Hat website. You can download a copy of the distribution by following the steps below.

#### Procedure

- Download the latest version of the Red Hat AMQ Streams archive from the [Customer Portal](#).

### 2.3. INSTALLING AMQ STREAMS

Follow this procedure to install the latest version of AMQ Streams on Red Hat Enterprise Linux.

For instructions on upgrading an existing cluster to AMQ Streams 1.4, see [AMQ Streams and Kafka upgrades](#).

#### Prerequisites

- Download the [installation archive](#).
- Review the [Section 1.3, "Supported Configurations"](#)

#### Procedure

1. Add new **kafka** user and group.

```
sudo groupadd kafka
sudo useradd -g kafka kafka
sudo passwd kafka
```

2. Create directory **/opt/kafka**.

```
sudo mkdir /opt/kafka
```

3. Create a temporary directory and extract the contents of the AMQ Streams ZIP file.

```
mkdir /tmp/kafka
unzip amq-streams_y.y-x.x.x.zip -d /tmp/kafka
```

4. Move the extracted contents into **/opt/kafka** directory and delete the temporary directory.

```
sudo mv /tmp/kafka/kafka_y.y-x.x.x/* /opt/kafka/
rm -r /tmp/kafka
```

5. Change the ownership of the **/opt/kafka** directory to the **kafka** user.

```
sudo chown -R kafka:kafka /opt/kafka
```

6. Create directory **/var/lib/zookeeper** for storing ZooKeeper data and set its ownership to the **kafka** user.

```
sudo mkdir /var/lib/zookeeper
sudo chown -R kafka:kafka /var/lib/zookeeper
```

7. Create directory **/var/lib/kafka** for storing Kafka data and set its ownership to the **kafka** user.

```
sudo mkdir /var/lib/kafka
sudo chown -R kafka:kafka /var/lib/kafka
```

## 2.4. DATA STORAGE CONSIDERATIONS

An efficient data storage infrastructure is essential to the optimal performance of AMQ Streams.

AMQ Streams requires block storage and works well with cloud-based block storage solutions, such as Amazon Elastic Block Store (EBS). The use of file storage is not recommended.

Choose local storage when possible. If local storage is not available, you can use a Storage Area Network (SAN) accessed by a protocol such as Fibre Channel or iSCSI.

### 2.4.1. Apache Kafka and ZooKeeper storage support

Use separate disks for Apache Kafka and ZooKeeper.

Kafka supports JBOD (Just a Bunch of Disks) storage, a data storage configuration of multiple disks or volumes. JBOD provides increased data storage for Kafka brokers. It can also improve performance.

Solid-state drives (SSDs), though not essential, can improve the performance of Kafka in large clusters where data is sent to and received from multiple topics asynchronously. SSDs are particularly effective with ZooKeeper, which requires fast, low latency data access.



#### NOTE

You do not need to provision replicated storage because Kafka and ZooKeeper both have built-in data replication.

### 2.4.2. File systems

It is recommended that you configure your storage system to use the XFS file system. AMQ Streams is also compatible with the ext4 file system, but this might require additional configuration for best results.

### Additional resources

- For more information about XFS, see [The XFS File System](#).

## 2.5. RUNNING A SINGLE NODE AMQ STREAMS CLUSTER

This procedure shows how to run a basic AMQ Streams cluster consisting of single ZooKeeper and single Apache Kafka node both running on the same host. The default configuration files are used for ZooKeeper and Kafka.



### WARNING

A single node AMQ Streams cluster does not provide reliability and high availability and is suitable only for development purposes.

### Prerequisites

- AMQ Streams is installed on the host

### Running the cluster

1. Edit the ZooKeeper configuration file `/opt/kafka/config/zookeeper.properties`. Set the **dataDir** option to `/var/lib/zookeeper/`.

```
dataDir=/var/lib/zookeeper/
```

2. Start ZooKeeper.

```
su - kafka
/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties
```

3. Make sure that Apache ZooKeeper is running.

```
jcmd | grep zookeeper
```

Returns:

```
number org.apache.zookeeper.server.quorum.QuorumPeerMain config/zookeeper.properties
```

4. Edit the Kafka configuration file `/opt/kafka/config/server.properties`. Set the **log.dirs** option to `/var/lib/kafka/`.

```
log.dirs=/var/lib/kafka/
```

5. Start Kafka.



```
su - kafka
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
```

6. Make sure that Kafka is running.

```
jcmd | grep kafka
```

Returns:

```
number kafka.Kafka config/server.properties
```

### Additional resources

- For more information about installing AMQ Streams, see [Section 2.3, “Installing AMQ Streams”](#).
- For more information about configuring AMQ Streams, see [Section 2.8, “Configuring AMQ Streams”](#).

## 2.6. USING THE CLUSTER

This procedure shows how to start the Kafka producer and consumer to send and receive messages.

### Prerequisites

- AMQ Streams is installed on the host
- ZooKeeper and Kafka are up and running

A topic is created automatically in this procedure. Alternatively, you can configure and create topics before using the cluster. For more information on creating and managing topics, see [Topics](#).

### Procedure

1. Start the Kafka console producer.

```
/opt/kafka/bin/kafka-console-producer.sh --broker-list <bootstrap-address> --topic <topic-name>
```

For example:

```
/opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic my-topic
```

2. Type your message into the console where the producer is running.
3. Press *Enter* to send.

When Kafka creates a new topic automatically, you might receive warning messages to say the topic does not exist:

```
WARN Error while fetching metadata with correlation id 39 :
{4-3-16-topic1=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
```

```
WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id
4 :
{my-topic=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
```

The warning messages should not reappear after you start sending messages to the new topic.

The **auto.create.topics.enable** configuration property [enables the automatic creation of topics](#).

4. Start the message receiver.

```
/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server <bootstrap-address> --topic
<topic-name> --from-beginning
```

For example:

```
/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic my-topic -
-from-beginning
```

5. Confirm that you see the incoming messages in the consumer console.
6. Press Ctrl+C to exit the Kafka console producer and consumer.

## 2.7. STOPPING THE AMQ STREAMS SERVICES

You can stop the Kafka and ZooKeeper services by running a script. All connections to the Kafka and ZooKeeper services will be terminated.

### Prerequisites

- AMQ Streams is installed on the host
- ZooKeeper and Kafka are up and running

### Procedure

1. Stop the Kafka broker.

```
su - kafka
/opt/kafka/bin/kafka-server-stop.sh
```

2. Confirm that the Kafka broker is stopped.

```
jcmd | grep kafka
```

3. Stop ZooKeeper.

```
su - kafka
/opt/kafka/bin/zookeeper-server-stop.sh
```

## 2.8. CONFIGURING AMQ STREAMS

### Prerequisites

## Prerequisites

- AMQ Streams is downloaded and installed on the host

## Procedure

1. Open ZooKeeper and Kafka broker configuration files in a text editor. The configuration files are located at :

### ZooKeeper

**`/opt/kafka/config/zookeeper.properties`**

### Kafka

**`/opt/kafka/config/server.properties`**

2. Edit the configuration options. The configuration files are in the Java properties format. Every configuration option should be on separate line in the following format:

```
| <option> = <value>
```

Lines starting with **#** or **!** will be treated as comments and will be ignored by AMQ Streams components.

```
| # This is a comment
```

Values can be split into multiple lines by using **\** directly before the newline / carriage return.

```
| sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \  
    username="bob" \  
    password="bobs-password";
```

3. Save the changes
4. Restart the ZooKeeper or Kafka broker
5. Repeat this procedure on all the nodes of the cluster.

## CHAPTER 3. CONFIGURING ZOOKEEPER

Kafka uses ZooKeeper to store configuration data and for cluster coordination. It is strongly recommended to run a cluster of replicated ZooKeeper instances.

### 3.1. BASIC CONFIGURATION

The most important ZooKeeper configuration options are:

#### **tickTime**

ZooKeeper's basic time unit in milliseconds. It is used for heartbeats and session timeouts. For example, minimum session timeout will be two ticks.

#### **dataDir**

The directory where ZooKeeper stores its transaction logs and snapshots of its in-memory database. This should be set to the `/var/lib/zookeeper/` directory that was created during installation.

#### **clientPort**

Port number where clients can connect. Defaults to **2181**.

An example ZooKeeper configuration file named `config/zookeeper.properties` is located in the AMQ Streams installation directory. It is recommended to place the `dataDir` directory on a separate disk device to minimize the latency in ZooKeeper.

ZooKeeper configuration file should be located in `/opt/kafka/config/zookeeper.properties`. A basic example of the configuration file can be found below. The configuration file has to be readable by the `kafka` user.

```
tickTime=2000
dataDir=/var/lib/zookeeper/
clientPort=2181
```

### 3.2. ZOOKEEPER CLUSTER CONFIGURATION

For reliable ZooKeeper service, you should deploy ZooKeeper in a cluster. Hence, for production use cases, you must run a cluster of replicated ZooKeeper instances. ZooKeeper clusters are also referred to as *ensembles*.

ZooKeeper clusters usually consist of an odd number of nodes. ZooKeeper requires that a majority of the nodes in the cluster are up and running. For example:

- In a cluster with three nodes, at least two of the nodes must be up and running. This means it can tolerate one node being down.
- In a cluster consisting of five nodes, at least three nodes must be available. This means it can tolerate two nodes being down.
- In a cluster consisting of seven nodes, at least four nodes must be available. This means it can tolerate three nodes being down.

Having more nodes in the ZooKeeper cluster delivers better resiliency and reliability of the whole cluster.

ZooKeeper can run in clusters with an even number of nodes. The additional node, however, does not increase the resiliency of the cluster. A cluster with four nodes requires at least three nodes to be available and can tolerate only one node being down. Therefore it has exactly the same resiliency as a

cluster with only three nodes.

Ideally, the different ZooKeeper nodes should be located in different data centers or network segments. Increasing the number of ZooKeeper nodes increases the workload spent on cluster synchronization. For most Kafka use cases, a ZooKeeper cluster with 3, 5 or 7 nodes should be sufficient.



### WARNING

A ZooKeeper cluster with 3 nodes can tolerate only 1 unavailable node. This means that if a cluster node crashes while you are doing maintenance on another node your ZooKeeper cluster will be unavailable.

Replicated ZooKeeper configuration supports all configuration options supported by the standalone configuration. Additional options are added for the clustering configuration:

#### **initLimit**

Amount of time to allow followers to connect and sync to the cluster leader. The time is specified as a number of ticks (see the [timeTick option](#) for more details).

#### **syncLimit**

Amount of time for which followers can be behind the leader. The time is specified as a number of ticks (see the [timeTick option](#) for more details).

#### **reconfigEnabled**

Enables or disables [dynamic reconfiguration](#). Must be enabled in order to add or remove servers to a ZooKeeper cluster.

#### **standaloneEnabled**

Enables or disables standalone mode, where ZooKeeper runs with only one server.

In addition to the options above, every configuration file should contain a list of servers which should be members of the ZooKeeper cluster. The server records should be specified in the format **server.id=hostname:port1:port2**, where:

#### **id**

The ID of the ZooKeeper cluster node.

#### **hostname**

The hostname or IP address where the node listens for connections.

#### **port1**

The port number used for intra-cluster communication.

#### **port2**

The port number used for leader election.

The following is an example configuration file of a ZooKeeper cluster with three nodes:

```
timeTick=2000
dataDir=/var/lib/zookeeper/
initLimit=5
syncLimit=2
```

```
reconfigEnabled=true
standaloneEnabled=false
```

```
server.1=172.17.0.1:2888:3888:participant;172.17.0.1:2181
server.2=172.17.0.2:2888:3888:participant;172.17.0.2:2181
server.3=172.17.0.3:2888:3888:participant;172.17.0.3:2181
```



## NOTE

In ZooKeeper 3.5.7, the [four letter word](#) commands must be added to the allow list before they can be used. For more information, see the [ZooKeeper documentation](#).

## myid files

Each node in the ZooKeeper cluster must be assigned a unique **ID**. Each node's **ID** must be configured in a **myid** file and stored in the **dataDir** folder, like `/var/lib/zookeeper/`. The **myid** files should contain only a single line with the written **ID** as text. The **ID** can be any integer from 1 to 255. You must manually create this file on each cluster node. Using this file, each ZooKeeper instance will use the configuration from the corresponding **server.** line in the configuration file to configure its listeners. It will also use all other **server.** lines to identify other cluster members.

In the above example, there are three nodes, so each one will have a different **myid** with values **1**, **2**, and **3** respectively.

## 3.3. RUNNING MULTI-NODE ZOOKEEPER CLUSTER

This procedure will show you how to configure and run ZooKeeper as a multi-node cluster.



## NOTE

In ZooKeeper 3.5.7, the [four letter word](#) commands must be added to the allow list before they can be used. For more information, see the [ZooKeeper documentation](#).

## Prerequisites

- AMQ Streams is installed on all hosts which will be used as ZooKeeper cluster nodes.

## Running the cluster

1. Create the **myid** file in `/var/lib/zookeeper/`. Enter ID **1** for the first ZooKeeper node, **2** for the second ZooKeeper node, and so on.

```
su - kafka
echo "<NodeID>" > /var/lib/zookeeper/myid
```

For example:

```
su - kafka
echo "1" > /var/lib/zookeeper/myid
```

2. Edit the ZooKeeper `/opt/kafka/config/zookeeper.properties` configuration file for the following:

- Set the option **dataDir** to `/var/lib/zookeeper/`.
- Configure the **initLimit** and **syncLimit** options.
- Configure the **reconfigEnabled** and **standaloneEnabled** options.
- Add a list of all ZooKeeper nodes. The list should include also the current node.

### Example configuration for a node of ZooKeeper cluster with five members

```
timeTick=2000
dataDir=/var/lib/zookeeper/
initLimit=5
syncLimit=2
reconfigEnabled=true
standaloneEnabled=false

server.1=172.17.0.1:2888:3888:participant;172.17.0.1:2181
server.2=172.17.0.2:2888:3888:participant;172.17.0.2:2181
server.3=172.17.0.3:2888:3888:participant;172.17.0.3:2181
server.4=172.17.0.4:2888:3888:participant;172.17.0.4:2181
server.5=172.17.0.5:2888:3888:participant;172.17.0.5:2181
```

3. Start ZooKeeper with the default configuration file.

```
su - kafka
/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties
```

4. Verify that ZooKeeper is running.

```
jcmd | grep zookeeper
```

5. Repeat this procedure on all the nodes of the cluster.
6. Once all nodes of the clusters are up and running, verify that all nodes are members of the cluster by sending a **stat** command to each of the nodes using **ncat** utility.

### Use ncat stat to check the node status

```
echo stat | ncat localhost 2181
```

In the output you should see information that the node is either **leader** or **follower**.

### Example output from the ncat command

```
ZooKeeper version: 3.4.13-2d71af4dbe22557fda74f9a9b4309b15a7487f03, built on
06/29/2018 00:39 GMT
Clients:
/0:0:0:0:0:0:1:59726[0](queued=0,recved=1,sent=0)

Latency min/avg/max: 0/0/0
Received: 2
Sent: 1
Connections: 1
```

```

Outstanding: 0
Zxid: 0x200000000
Mode: follower
Node count: 4

```

### Additional resources

- For more information about installing AMQ Streams, see [Section 2.3, “Installing AMQ Streams”](#).
- For more information about configuring AMQ Streams, see [Section 2.8, “Configuring AMQ Streams”](#).

## 3.4. AUTHENTICATION

By default, ZooKeeper does not use any form of authentication and allows anonymous connections. However, it supports Java Authentication and Authorization Service (JAAS) which can be used to set up authentication using Simple Authentication and Security Layer (SASL). ZooKeeper supports authentication using the DIGEST-MD5 SASL mechanism with locally stored credentials.

### 3.4.1. Authentication with SASL

JAAS is configured using a separate configuration file. It is recommended to place the JAAS configuration file in the same directory as the ZooKeeper configuration (`/opt/kafka/config/`). The recommended file name is **zookeeper-jaas.conf**. When using a ZooKeeper cluster with multiple nodes, the JAAS configuration file has to be created on all cluster nodes.

JAAS is configured using contexts. Separate parts such as the server and client are always configured with a separate *context*. The context is a *configuration* option and has the following format:

```

ContextName {
    param1
    param2;
};

```

SASL Authentication is configured separately for server-to-server communication (communication between ZooKeeper instances) and client-to-server communication (communication between Kafka and ZooKeeper). Server-to-server authentication is relevant only for ZooKeeper clusters with multiple nodes.

#### Server-to-Server authentication

For server-to-server authentication, the JAAS configuration file contains two parts:

- The server configuration
- The client configuration

When using DIGEST-MD5 SASL mechanism, the **QuorumServer** context is used to configure the authentication server. It must contain all the usernames to be allowed to connect together with their passwords in an unencrypted form. The second context, **QuorumLearner**, has to be configured for the client which is built into ZooKeeper. It also contains the password in an unencrypted form. An example of the JAAS configuration file for DIGEST-MD5 mechanism can be found below:

```

QuorumServer {
    org.apache.zookeeper.server.auth.DigestLoginModule required

```



```

    user_zookeeper="123456";
};

QuorumLearner {
    org.apache.zookeeper.server.auth.DigestLoginModule required
    username="zookeeper"
    password="123456";
};

```

In addition to the JAAS configuration file, you must enable the server-to-server authentication in the regular ZooKeeper configuration file by specifying the following options:

```

quorum.auth.enableSasl=true
quorum.auth.learnerRequireSasl=true
quorum.auth.serverRequireSasl=true
quorum.auth.learner.loginContext=QuorumLearner
quorum.auth.server.loginContext=QuorumServer
quorum.cnxn.threads.size=20

```

Use the **KAFKA\_OPTS** environment variable to pass the JAAS configuration file to the ZooKeeper server as a Java property:

```

su - kafka
export KAFKA_OPTS="-Djava.security.auth.login.config=/opt/kafka/config/zookeeper-jaas.conf";
/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties

```

For more information about server-to-server authentication, see [ZooKeeper wiki](#).

### Client-to-Server authentication

Client-to-server authentication is configured in the same JAAS file as the server-to-server authentication. However, unlike the server-to-server authentication, it contains only the server configuration. The client part of the configuration has to be done in the client. For information on how to configure a Kafka broker to connect to ZooKeeper using authentication, see the [Kafka installation](#) section.

Add the Server context to the JAAS configuration file to configure client-to-server authentication. For DIGEST-MD5 mechanism it configures all usernames and passwords:

```

Server {
    org.apache.zookeeper.server.auth.DigestLoginModule required
    user_super="123456"
    user_kafka="123456"
    user_someoneelse="123456";
};

```

After configuring the JAAS context, enable the client-to-server authentication in the ZooKeeper configuration file by adding the following line:

```

requireClientAuthScheme=sasl
authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
authProvider.2=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
authProvider.3=org.apache.zookeeper.server.auth.SASLAuthenticationProvider

```

You must add the **authProvider.<ID>** property for every server that is part of the ZooKeeper cluster.

Use the **KAFKA\_OPTS** environment variable to pass the JAAS configuration file to the ZooKeeper server as a Java property:

```
su - kafka
export KAFKA_OPTS="-Djava.security.auth.login.config=/opt/kafka/config/zookeeper-jaas.conf";
/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties
```

For more information about configuring ZooKeeper authentication in Kafka brokers, see [Section 4.6, "ZooKeeper authentication"](#).

### 3.4.2. Enabling Server-to-server authentication using DIGEST-MD5

This procedure describes how to enable authentication using the SASL DIGEST-MD5 mechanism between the nodes of the ZooKeeper cluster.

#### Prerequisites

- AMQ Streams is installed on the host
- ZooKeeper cluster is [configured](#) with multiple nodes.

#### Enabling SASL DIGEST-MD5 authentication

1. On all ZooKeeper nodes, create or edit the **/opt/kafka/config/zookeeper-jaas.conf** JAAS configuration file and add the following contexts:

```
QuorumServer {
    org.apache.zookeeper.server.auth.DigestLoginModule required
    user_<Username>=<Password>;
};

QuorumLearner {
    org.apache.zookeeper.server.auth.DigestLoginModule required
    username=<Username>
    password=<Password>;
};
```

The username and password must be the same in both JAAS contexts. For example:

```
QuorumServer {
    org.apache.zookeeper.server.auth.DigestLoginModule required
    user_zookeeper="123456";
};

QuorumLearner {
    org.apache.zookeeper.server.auth.DigestLoginModule required
    username="zookeeper"
    password="123456";
};
```

2. On all ZooKeeper nodes, edit the **/opt/kafka/config/zookeeper.properties** ZooKeeper configuration file and set the following options:

```
quorum.auth.enableSasl=true
```

```

quorum.auth.learnerRequireSasl=true
quorum.auth.serverRequireSasl=true
quorum.auth.learner.loginContext=QuorumLearner
quorum.auth.server.loginContext=QuorumServer
quorum.cnxn.threads.size=20

```

- Restart all ZooKeeper nodes one by one. To pass the JAAS configuration to ZooKeeper, use the **KAFKA\_OPTS** environment variable.

```

su - kafka
export KAFKA_OPTS="-Djava.security.auth.login.config=/opt/kafka/config/zookeeper-jaas.conf"; /opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties

```

### Additional resources

- For more information about installing AMQ Streams, see [Section 2.3, “Installing AMQ Streams”](#).
- For more information about configuring AMQ Streams, see [Section 2.8, “Configuring AMQ Streams”](#).
- For more information about running a ZooKeeper cluster, see [Section 3.3, “Running multi-node ZooKeeper cluster”](#).

### 3.4.3. Enabling Client-to-server authentication using DIGEST-MD5

This procedure describes how to enable authentication using the SASL DIGEST-MD5 mechanism between ZooKeeper clients and ZooKeeper.

#### Prerequisites

- AMQ Streams is installed on the host
- ZooKeeper cluster is [configured and running](#).

#### Enabling SASL DIGEST-MD5 authentication

- On all ZooKeeper nodes, create or edit the **/opt/kafka/config/zookeeper-jaas.conf** JAAS configuration file and add the following context:

```

Server {
  org.apache.zookeeper.server.auth.DigestLoginModule required
  user_super="<SuperUserPassword>"
  user<Username1>_="<Password1>" user<Username2>_="<Password2>";
};

```

The **super** automatically has administrator privileges. The file can contain multiple users, but only one additional user is required by the Kafka brokers. The recommended name for the Kafka user is **kafka**.

The following example shows the **Server** context for client-to-server authentication:

```

Server {
  org.apache.zookeeper.server.auth.DigestLoginModule required

```

```

user_super="123456"
user_kafka="123456";
};

```

- On all ZooKeeper nodes, edit the `/opt/kafka/config/zookeeper.properties` ZooKeeper configuration file and set the following options:

```

requireClientAuthScheme=sasl
authProvider.<IdOfBroker1>=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
authProvider.<IdOfBroker2>=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
authProvider.<IdOfBroker3>=org.apache.zookeeper.server.auth.SASLAuthenticationProvider

```

The `authProvider.<ID>` property has to be added for every node which is part of the ZooKeeper cluster. An example three-node ZooKeeper cluster configuration must look like the following:

```

requireClientAuthScheme=sasl
authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
authProvider.2=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
authProvider.3=org.apache.zookeeper.server.auth.SASLAuthenticationProvider

```

- Restart all ZooKeeper nodes one by one. To pass the JAAS configuration to ZooKeeper, use the `KAFKA_OPTS` environment variable.

```

su - kafka
export KAFKA_OPTS="-Djava.security.auth.login.config=/opt/kafka/config/zookeeper-jaas.conf"; /opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties

```

### Additional resources

- For more information about installing AMQ Streams, see [Section 2.3, “Installing AMQ Streams”](#).
- For more information about configuring AMQ Streams, see [Section 2.8, “Configuring AMQ Streams”](#).
- For more information about running a ZooKeeper cluster, see [Section 3.3, “Running multi-node ZooKeeper cluster”](#).

## 3.5. AUTHORIZATION

ZooKeeper supports access control lists (ACLs) to protect data stored inside it. Kafka brokers can automatically configure the ACL rights for all ZooKeeper records they create so no other ZooKeeper user can modify them.

For more information about enabling ZooKeeper ACLs in Kafka brokers, see [Section 4.8, “ZooKeeper authorization”](#).

## 3.6. TLS

The version of ZooKeeper which is part of AMQ Streams currently does not support TLS for encryption or authentication.

## 3.7. ADDITIONAL CONFIGURATION OPTIONS

You can set the following additional ZooKeeper configuration options based on your use case:

### **maxClientCnxns**

The maximum number of concurrent client connections to a single member of the ZooKeeper cluster.

### **autopurge.snapRetainCount**

Number of snapshots of ZooKeeper's in-memory database which will be retained. Default value is **3**.

### **autopurge.purgeInterval**

The time interval in hours for purging snapshots. The default value is **0** and this option is disabled.

All available configuration options can be found in the [ZooKeeper documentation](#).

## 3.8. LOGGING

ZooKeeper is using *log4j* as their logging infrastructure. Logging configuration is by default read from the **log4j.properties** configuration file which should be placed either in the `/opt/kafka/config/` directory or in the classpath. The location and name of the configuration file can be changed using the Java property **log4j.configuration** which can be passed to ZooKeeper using the **KAFKA\_LOG4J\_OPTS** environment variable:

```
su - kafka
export KAFKA_LOG4J_OPTS="-Dlog4j.configuration=file:/my/path/to/log4j.properties";
/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties
```

For more information about Log4j configurations, see [Log4j documentation](#).

## CHAPTER 4. CONFIGURING KAFKA

Kafka uses a properties file to store static configuration. The recommended location for the configuration file is `/opt/kafka/config/server.properties`. The configuration file has to be readable by the **kafka** user.

AMQ Streams ships an example configuration file that highlights various basic and advanced features of the product. It can be found under **config/server.properties** in the AMQ Streams installation directory.

This chapter explains the most important configuration options. For a complete list of supported Kafka broker configuration options, see [Appendix A, Broker configuration parameters](#).

### 4.1. ZOOKEEPER

Kafka brokers need ZooKeeper to store some parts of their configuration as well as to coordinate the cluster (for example to decide which node is a leader for which partition). Connection details for the ZooKeeper cluster are stored in the configuration file. The field **zookeeper.connect** contains a comma-separated list of hostnames and ports of members of the zookeeper cluster.

For example:

```
zookeeper.connect=zoo1.my-domain.com:2181,zoo2.my-domain.com:2181,zoo3.my-domain.com:2181
```

Kafka will use these addresses to connect to the ZooKeeper cluster. With this configuration, all Kafka **znodes** will be created directly in the root of ZooKeeper database. Therefore, such a ZooKeeper cluster could be used only for a single Kafka cluster. To configure multiple Kafka clusters to use single ZooKeeper cluster, specify a base (prefix) path at the end of the ZooKeeper connection string in the Kafka configuration file:

```
zookeeper.connect=zoo1.my-domain.com:2181,zoo2.my-domain.com:2181,zoo3.my-domain.com:2181/my-cluster-1
```

### 4.2. LISTENERS

Kafka brokers can be configured to use multiple listeners. Each listener can be used to listen on a different port or network interface and can have different configuration. Listeners are configured in the **listeners** property in the configuration file. The **listeners** property contains a list of listeners with each listener configured as `<listenerName>://<hostname>:<port>`. When the hostname value is empty, Kafka will use `java.net.InetAddress.getCanonicalHostName()` as hostname. The following example shows how multiple listeners might be configured:

```
listeners=INT1://:9092,INT2://:9093,REPLICATION://:9094
```

When a Kafka client wants to connect to a Kafka cluster, it first connects to a *bootstrap server*. The *bootstrap server* is one of the cluster nodes. It will provide the client with a list of all other brokers which are part of the cluster and the client will connect to them individually. By default the *bootstrap server* will provide the client with a list of nodes based on the **listeners** field.

#### Advertised listeners

It is possible to give the client a different set of addresses than given in the listeners property. It is useful in situations when additional network infrastructure, such as a proxy, is between the client and the broker, or when an external DNS name should be used instead of an IP address. Here, the broker allows

defining the advertised addresses of the listeners in the `advertised.listeners` configuration property. This property has the same format as the `listeners` property. The following example shows how to configure advertised listeners:

```
listeners=INT1://:9092,INT2://:9093
advertised.listeners=INT1://my-broker-1.my-domain.com:1234,INT2://my-broker-1.my-domain.com:1234:9093
```



#### NOTE

The names of the listeners have to match the names of the listeners from the **listeners** property.

### Inter-broker listeners

When the cluster has replicated topics, the brokers responsible for such topics need to communicate with each other in order to replicate the messages in those topics. When multiple listeners are configured, the configuration field **inter.broker.listener.name** can be used to specify the name of the listener which should be used for replication between brokers. For example:

```
inter.broker.listener.name=REPLICATION
```

## 4.3. COMMIT LOGS

Apache Kafka stores all records it receives from producers in commit logs. The commit logs contain the actual data, in the form of records, that Kafka needs to deliver. These are not the application log files which record what the broker is doing.

### Log directories

You can configure log directories using the **log.dirs** property file to store commit logs in one or multiple log directories. It should be set to `/var/lib/kafka` directory created during installation:

```
log.dirs=/var/lib/kafka
```

For performance reasons, you can configure `log.dirs` to multiple directories and place each of them on a different physical device to improve disk I/O performance. For example:

```
log.dirs=/var/lib/kafka1,/var/lib/kafka2,/var/lib/kafka3
```

## 4.4. BROKER ID

Broker ID is a unique identifier for each broker in the cluster. You can assign an integer greater than or equal to 0 as broker ID. The broker ID is used to identify the brokers after restarts or crashes and it is therefore important that the id is stable and does not change over time. The broker ID is configured in the broker properties file:

```
broker.id=1
```

## 4.5. RUNNING A MULTI-NODE KAFKA CLUSTER

This procedure describes how to configure and run Kafka as a multi-node cluster.

## Prerequisites

- AMQ Streams is [installed](#) on all hosts which will be used as Kafka brokers.
- A ZooKeeper cluster is [configured and running](#).

## Running the cluster

For each Kafka broker in your AMQ Streams cluster:

1. Edit the `/opt/kafka/config/server.properties` Kafka configuration file as follows:
  - Set the **broker.id** field to **0** for the first broker, **1** for the second broker, and so on.
  - Configure the details for connecting to ZooKeeper in the **zookeeper.connect** option.
  - Configure the Kafka listeners.
  - Set the directories where the commit logs should be stored in the **logs.dir** directory.Here we see an example configuration for a Kafka broker:

```
broker.id=0
zookeeper.connect=zoo1.my-domain.com:2181,zoo2.my-domain.com:2181,zoo3.my-
domain.com:2181
listeners=REPLICATION://:9091,PLAINTEXT://:9092
inter.broker.listener.name=REPLICATION
log.dirs=/var/lib/kafka
```

In a typical installation where each Kafka broker is running on identical hardware, only the **broker.id** configuration property will differ between each broker config.

2. Start the Kafka broker with the default configuration file.

```
su - kafka
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
```

3. Verify that the Kafka broker is running.

```
jcmd | grep Kafka
```

## Verifying the brokers

Once all nodes of the clusters are up and running, verify that all nodes are members of the Kafka cluster by sending a **dump** command to one of the ZooKeeper nodes using the **ncat** utility. The command prints all Kafka brokers registered in ZooKeeper.

1. Use `ncat stat` to check the node status.

```
echo dump | ncat zoo1.my-domain.com 2181
```

The output should contain all Kafka brokers you just configured and started.

Example output from the **ncat** command for Kafka cluster with 3 nodes:



```

SessionTracker dump:
org.apache.zookeeper.server.quorum.LearnerSessionTracker@28848ab9
ephemeral nodes dump:
Sessions with Ephemerals (3):
0x20000015dd00000:
    /brokers/ids/1
0x10000015dc70000:
    /controller
    /brokers/ids/0
0x10000015dc70001:
    /brokers/ids/2

```

### Additional resources

- For more information about installing AMQ Streams, see [Section 2.3, “Installing AMQ Streams”](#).
- For more information about configuring AMQ Streams, see [Section 2.8, “Configuring AMQ Streams”](#).
- For more information about running a ZooKeeper cluster, see [Section 3.3, “Running multi-node ZooKeeper cluster”](#).
- For a complete list of supported Kafka broker configuration options, see [Appendix A, \*Broker configuration parameters\*](#).

## 4.6. ZOOKEEPER AUTHENTICATION

By default, connections between ZooKeeper and Kafka are not authenticated. However, Kafka and ZooKeeper support Java Authentication and Authorization Service (JAAS) which can be used to set up authentication using Simple Authentication and Security Layer (SASL). ZooKeeper supports authentication using the DIGEST-MD5 SASL mechanism with locally stored credentials.

### 4.6.1. JAAS Configuration

SASL authentication for ZooKeeper connections has to be configured in the JAAS configuration file. By default, Kafka will use the JAAS context named **Client** for connecting to ZooKeeper. The **Client** context should be configured in the `/opt/kafka/config/jass.conf` file. The context has to enable the **PLAIN** SASL authentication, as in the following example:

```

Client {
    org.apache.kafka.common.security.plain.PlainLoginModule required
    username="kafka"
    password="123456";
};

```

### 4.6.2. Enabling ZooKeeper authentication

This procedure describes how to enable authentication using the SASL DIGEST-MD5 mechanism when connecting to ZooKeeper.

#### Prerequisites

- Client-to-server authentication is [enabled](#) in ZooKeeper

## Enabling SASL DIGEST-MD5 authentication

1. On all Kafka broker nodes, create or edit the `/opt/kafka/config/jaas.conf` JAAS configuration file and add the following context:

```
Client {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="<Username>"
  password="<Password>";
};
```

The username and password should be the same as configured in ZooKeeper.

Following example shows the **Client** context:

```
Client {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="kafka"
  password="123456";
};
```

2. Restart all Kafka broker nodes one by one. To pass the JAAS configuration to Kafka brokers, use the **KAFKA\_OPTS** environment variable.

```
su - kafka
export KAFKA_OPTS="-Djava.security.auth.login.config=/opt/kafka/config/jaas.conf";
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
```

### Additional resources

- For more information about configuring client-to-server authentication in ZooKeeper, see [Section 3.4, "Authentication"](#).

## 4.7. AUTHORIZATION

Authorization in Kafka brokers is implemented using authorizer plugins.

In this section we describe how to use **SimpleAclAuthorizer**, the authorizer plugin provided with Kafka.

Alternatively, you can use your own authorization plugins. For example, if you are using [OAuth 2.0 token-based authentication](#), you can use [OAuth 2.0 authorization](#).

### 4.7.1. Simple ACL authorizer

Authorizer plugins, including **SimpleAclAuthorizer**, are enabled through the **authorizer.class.name** property:

```
authorizer.class.name=kafka.security.auth.SimpleAclAuthorizer
```

A fully-qualified name is required for the chosen authorizer. For **SimpleAclAuthorizer**, the fully-qualified name is **kafka.security.auth.SimpleAclAuthorizer**.

#### 4.7.1.1. ACL rules

**SimpleAclAuthorizer** uses ACL rules to manage access to Kafka brokers.

ACL rules are defined in the format:

Principal **P** is allowed / denied operation **O** on Kafka resource **R** from host **H**

For example, a rule might be set so that user:

**John** can **view** the topic **comments** from host **127.0.0.1**

Host is the IP address of the machine that John is connecting from.

In most cases, the user is a producer or consumer application:

**Consumer01** can **write** to the consumer group **accounts** from host **127.0.0.1**

### If ACL rules are not present

If ACL rules are not present for a given resource, all actions are denied. This behavior can be changed by setting the property **allow.everyone.if.no.acl.found** to **true** in the Kafka configuration file **/opt/kafka/config/server.properties**.

#### 4.7.1.2. Principals

A *principal* represents the identity of a user. The format of the ID depends on the authentication mechanism used by clients to connect to Kafka:

- **User:ANONYMOUS** when connected without authentication.
- **User:<username>** when connected using simple authentication mechanisms, such as PLAIN or SCRAM.  
For example **User:admin** or **User:user1**.
- **User:<DistinguishedName>** when connected using TLS client authentication.  
For example **User:CN=user1,O=MyCompany,L=Prague,C=CZ**.
- **User:<Kerberos username>** when connected using Kerberos.

The *DistinguishedName* is the distinguished name from the client certificate.

The *Kerberos username* is the primary part of the Kerberos principal, which is used by default when connecting using Kerberos. You can use the **sasl.kerberos.principal.to.local.rules** property to configure how the Kafka principal is built from the Kerberos principal.

#### 4.7.1.3. Authentication of users

To use authorization, you need to have authentication enabled and used by your clients. Otherwise, all connections will have the principal **User:ANONYMOUS**.

For more information on methods of authentication, see [Encryption and authentication](#).

#### 4.7.1.4. Super users

Super users are allowed to take all actions regardless of the ACL rules.

Super users are defined in the Kafka configuration file using the property **super.users**.

For example:

```
super.users=User:admin,User:operator
```

#### 4.7.1.5. Replica broker authentication

When authorization is enabled, it is applied to all listeners and all connections. This includes the inter-broker connections used for replication of data between brokers. If enabling authorization, therefore, ensure that you use authentication for inter-broker connections and give the users used by the brokers sufficient rights. For example, if authentication between brokers uses the **kafka-broker** user, then super user configuration must include the username **super.users=User:kafka-broker**.

#### 4.7.1.6. Supported resources

You can apply Kafka ACLs to these types of resource:

- Topics
- Consumer groups
- The cluster
- TransactionId
- DelegationToken

#### 4.7.1.7. Supported operations

**SimpleAclAuthorizer** authorizes operations on resources.

Fields with **X** in the following table mark the supported operations for each resource.

	Topics	Consumer Groups	Cluster
Read	X	X	
Write	X		
Create			X
Delete	X		
Alter	X		
Describe	X	X	X
ClusterAction			X
All	X	X	X

### 4.7.1.8. ACL management options

ACL rules are managed using the **bin/kafka-acls.sh** utility, which is provided as part of the Kafka distribution package.

Use **kafka-acls.sh** parameter options to add, list and remove ACL rules, and perform other functions.

The parameters require a double-hyphen convention, such as **--add**.

Option	Type	Description	Default
<b>add</b>	Action	Add ACL rule.	
<b>remove</b>	Action	Remove ACL rule.	
<b>list</b>	Action	List ACL rules.	
<b>authorizer</b>	Action	Fully-qualified class name of the authorizer.	<b>kafka.security.auth.S impleAclAuthorizer</b>
<b>authorizer-properties</b>	Configuration	Key/value pairs passed to the authorizer for initialization.  For <b>SimpleAclAuthorizer</b> , the example values are: <b>zookeeper.connect=zoo1.my-domain.com:2181</b> .	
<b>bootstrap-server</b>	Resource	Host/port pairs to connect to the Kafka cluster.	Use this option or the <b>authorizer</b> option, not both.
<b>command-config</b>	Resource	Configuration property file to pass to the Admin Client, which is used in conjunction with the <b>bootstrap-server</b> parameter.	
<b>cluster</b>	Resource	Specifies a cluster as an ACL resource.	

Option	Type	Description	Default
<b>topic</b>	Resource	<p>Specifies a topic name as an ACL resource.</p> <p>An asterisk (*) used as a wildcard translates to <i>all topics</i>.</p> <p>A single command can specify multiple <b>--topic</b> options.</p>	
<b>group</b>	Resource	<p>Specifies a consumer group name as an ACL resource.</p> <p>A single command can specify multiple <b>--group</b> options.</p>	
<b>transactional-id</b>	Resource	<p>Specifies a transactional ID as an ACL resource.</p> <p>Transactional delivery means that all messages sent by a producer to multiple partitions must be successfully delivered or none of them.</p> <p>An asterisk (*) used as a wildcard translates to <i>all IDs</i>.</p>	
<b>delegation-token</b>	Resource	<p>Specifies a delegation token as an ACL resource.</p> <p>An asterisk (*) used as a wildcard translates to <i>all tokens</i>.</p>	

Option	Type	Description	Default
<b>resource-pattern-type</b>	Configuration	<p>Specifies a type of resource pattern for the <b>add</b> parameter or a resource pattern filter value for the <b>list</b> or <b>remove</b> parameters.</p> <p>Use <b>literal</b> or <b>prefixed</b> as the resource pattern type for a resource name.</p> <p>Use <b>any</b> or <b>match</b> as resource pattern filter values, or a specific pattern type filter.</p>	<b>literal</b>
<b>allow-principal</b>	Principal	<p>Principal added to an allow ACL rule.</p> <p>A single command can specify multiple <b>--allow-principal</b> options.</p>	
<b>deny-principal</b>	Principal	<p>Principal added to a deny ACL rule.</p> <p>A single command can specify multiple <b>--deny-principal</b> options.</p>	
<b>principal</b>	Principal	<p>Principal name used with the <b>list</b> parameter to return a list of ACLs for the principal.</p> <p>A single command can specify multiple <b>--principal</b> options.</p>	
<b>allow-host</b>	Host	<p>IP address that allows access to the principals listed in <b>--allow-principal</b>.</p> <p>Hostnames or CIDR ranges are not supported.</p>	If <b>--allow-principal</b> is specified, defaults to <b>*</b> meaning "all hosts".

Option	Type	Description	Default
<b>deny-host</b>	Host	<p>IP address that denies access to the principals listed in <b>--deny-principal</b>.</p> <p>Hostnames or CIDR ranges are not supported.</p>	if <b>--deny-principal</b> is specified, defaults to * meaning "all hosts".
<b>operation</b>	Operation	<p>Allows or denies an operation.</p> <p>A single command can specify multiple <b>--operation</b> options can be specified in single command.</p>	All
<b>producer</b>	Shortcut	A shortcut to allow or deny all operations needed by a message producer (WRITE and DESCRIBE on topic, CREATE on cluster).	
<b>consumer</b>	Shortcut	A shortcut to allow or deny all operations needed by a message consumer (READ and DESCRIBE on topic, READ on consumer group).	
<b>idempotent</b>	Shortcut	<p>A shortcut to enable idempotence when used with the <b>--producer</b> parameter, so that messages are delivered exactly once to a partition.</p> <p>Idempotence is enabled automatically if the producer is authorized to send messages based on a specific transactional ID.</p>	
<b>force</b>	Shortcut	A shortcut to accept all queries and do not prompt.	



## 4.7.2. Enabling authorization

This procedure describes how to enable the **SimpleAclAuthorizer** plugin for authorization in Kafka brokers.

### Prerequisites

- [AMQ Streams is installed](#) on all hosts used as Kafka brokers.

### Procedure

1. Edit the `/opt/kafka/config/server.properties` Kafka configuration file to use the **SimpleAclAuthorizer**.

```
authorizer.class.name=kafka.security.auth.SimpleAclAuthorizer
```

2. (Re)start the Kafka brokers.

### Additional resources

- For more information about configuring AMQ Streams, see [Section 2.8, “Configuring AMQ Streams”](#).
- For more information about running a Kafka cluster, see [Section 4.5, “Running a multi-node Kafka cluster”](#).

## 4.7.3. Adding ACL rules

**SimpleAclAuthorizer** uses Access Control Lists (ACLs), which define a set of rules describing what users can and cannot do.

This procedure describes how to add ACL rules when using the **SimpleAclAuthorizer** plugin in Kafka brokers.

Rules are added using the **kafka-acls.sh** utility and stored in ZooKeeper.

### Prerequisites

- [AMQ Streams is installed](#) on all hosts used as Kafka brokers.
- Authorization is [enabled](#) in Kafka brokers.

### Procedure

1. Run **kafka-acls.sh** with the **--add** option.

Examples:

- Allow **user1** and **user2** access to read from **myTopic** using the **MyConsumerGroup** consumer group.

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=zoo1.my-domain.com:2181
--add --operation Read --topic myTopic --allow-principal User:user1 --allow-principal
User:user2
```

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=zoo1.my-domain.com:2181
--add --operation Describe --topic myTopic --allow-principal User:user1 --allow-principal
User:user2
```

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=zoo1.my-domain.com:2181
--add --operation Read --operation Describe --group MyConsumerGroup --allow-
principal User:user1 --allow-principal User:user2
```

- Deny **user1** access to read **myTopic** from IP address host **127.0.0.1**.

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=zoo1.my-domain.com:2181
--add --operation Describe --operation Read --topic myTopic --group MyConsumerGroup
--deny-principal User:user1 --deny-host 127.0.0.1
```

- Add **user1** as the consumer of **myTopic** with **MyConsumerGroup**.

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=zoo1.my-domain.com:2181
--add --consumer --topic myTopic --group MyConsumerGroup --allow-principal
User:user1
```

### Additional resources

- For a list of all **kafka-acls.sh** options, see [Section 4.7.1, “Simple ACL authorizer”](#).

### 4.7.4. Listing ACL rules

This procedure describes how to list existing ACL rules when using the **SimpleAclAuthorizer** plugin in Kafka brokers.

Rules are listed using the **kafka-acls.sh** utility.

### Prerequisites

- [AMQ Streams is installed](#) on all hosts used as Kafka brokers.
- Authorization is [enabled](#) in Kafka brokers
- ACLs have been [added](#).

### Procedure

- Run **kafka-acls.sh** with the **--list** option.  
For example:

```
$ bin/kafka-acls.sh --authorizer-properties zookeeper.connect=zoo1.my-domain.com:2181 --
list --topic myTopic
```

```
Current ACLs for resource `Topic:myTopic`:
```

```
User:user1 has Allow permission for operations: Read from hosts: *
User:user2 has Allow permission for operations: Read from hosts: *
User:user2 has Deny permission for operations: Read from hosts: 127.0.0.1
```

```
User:user1 has Allow permission for operations: Describe from hosts: *
User:user2 has Allow permission for operations: Describe from hosts: *
User:user2 has Deny permission for operations: Describe from hosts: 127.0.0.1
```

### Additional resources

- For a list of all **kafka-acls.sh** options, see [Section 4.7.1, “Simple ACL authorizer”](#).

## 4.7.5. Removing ACL rules

This procedure describes how to remove ACL rules when using the **SimpleAclAuthorizer** plugin in Kafka brokers.

Rules are removed using the **kafka-acls.sh** utility.

### Prerequisites

- [AMQ Streams is installed](#) on all hosts used as Kafka brokers.
- Authorization is [enabled](#) in Kafka brokers.
- ACLs have been [added](#).

### Procedure

- Run **kafka-acls.sh** with the **--remove** option.

Examples:

- Remove the ACL allowing Allow **user1** and **user2** access to read from **myTopic** using the **MyConsumerGroup** consumer group.

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=zoo1.my-domain.com:2181 --
remove --operation Read --topic myTopic --allow-principal User:user1 --allow-principal
User:user2
```

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=zoo1.my-domain.com:2181 --
remove --operation Describe --topic myTopic --allow-principal User:user1 --allow-principal
User:user2
```

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=zoo1.my-domain.com:2181 --
remove --operation Read --operation Describe --group MyConsumerGroup --allow-principal
User:user1 --allow-principal User:user2
```

- Remove the ACL adding **user1** as the consumer of **myTopic** with **MyConsumerGroup**.

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=zoo1.my-domain.com:2181 --
remove --consumer --topic myTopic --group MyConsumerGroup --allow-principal User:user1
```

- Remove the ACL denying **user1** access to read **myTopic** from IP address host **127.0.0.1**.

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=zoo1.my-domain.com:2181 --
remove --operation Describe --operation Read --topic myTopic --group MyConsumerGroup -
-deny-principal User:user1 --deny-host 127.0.0.1
```

## Additional resources

- For a list of all **kafka-acls.sh** options, see [Section 4.7.1, “Simple ACL authorizer”](#).
- For more information about enabling authorization, see [Section 4.7.2, “Enabling authorization”](#).

## 4.8. ZOOKEEPER AUTHORIZATION

When authentication is enabled between Kafka and ZooKeeper, you can use ZooKeeper Access Control List (ACL) rules to automatically control access to Kafka’s metadata stored in ZooKeeper.

### 4.8.1. ACL Configuration

Enforcement of ZooKeeper ACL rules is controlled by the **zookeeper.set.acl** property in the **config/server.properties** Kafka configuration file.

The property is disabled by default and enabled by setting to **true**:

```
zookeeper.set.acl=true
```

If ACL rules are enabled, when a **znode** is created in ZooKeeper only the Kafka user who created it can modify or delete it. All other users have read-only access.

Kafka sets ACL rules only for newly created ZooKeeper **znodes**. If the ACLs are only enabled after the first start of the cluster, the **zookeeper-security-migration.sh** tool can set ACLs on all existing **znodes**.

### Confidentiality of data in ZooKeeper

Data stored in ZooKeeper includes:

- Topic names and their configuration
- Salted and hashed user credentials when SASL SCRAM authentication is used.

But ZooKeeper does not store any records sent and received using Kafka. The data stored in ZooKeeper is assumed to be non-confidential.

If the data is to be regarded as confidential (for example because topic names contain customer IDs), the only option available for protection is isolating ZooKeeper on the network level and allowing access only to Kafka brokers.

### 4.8.2. Enabling ZooKeeper ACLs for a new Kafka cluster

This procedure describes how to enable ZooKeeper ACLs in Kafka configuration for a new Kafka cluster. Use this procedure only before the first start of the Kafka cluster. For enabling ZooKeeper ACLs in a cluster that is already running, see [Section 4.8.3, “Enabling ZooKeeper ACLs in an existing Kafka cluster”](#).

#### Prerequisites

- AMQ Streams is [installed](#) on all hosts which will be used as Kafka brokers.
- ZooKeeper cluster is [configured and running](#).
- Client-to-server authentication is [enabled](#) in ZooKeeper.
- ZooKeeper authentication is [enabled](#) in the Kafka brokers.

- Kafka brokers have not yet been started.

### Procedure

1. Edit the `/opt/kafka/config/server.properties` Kafka configuration file to set the `zookeeper.set.acl` field to `true` on all cluster nodes.

```
zookeeper.set.acl=true
```

2. Start the Kafka brokers.

### 4.8.3. Enabling ZooKeeper ACLs in an existing Kafka cluster

This procedure describes how to enable ZooKeeper ACLs in Kafka configuration for a Kafka cluster that is running. Use the `zookeeper-security-migration.sh` tool to set ZooKeeper ACLs on all existing **znodes**. The `zookeeper-security-migration.sh` is available as part of AMQ Streams, and can be found in the `bin` directory.

#### Prerequisites

- Kafka cluster is [configured and running](#).

#### Enabling the ZooKeeper ACLs

1. Edit the `/opt/kafka/config/server.properties` Kafka configuration file to set the `zookeeper.set.acl` field to `true` on all cluster nodes.

```
zookeeper.set.acl=true
```

2. Restart all Kafka brokers one by one.
3. Set the ACLs on all existing ZooKeeper **znodes** using the `zookeeper-security-migration.sh` tool.

```
su - kafka
cd /opt/kafka
KAFKA_OPTS="-Djava.security.auth.login.config=./config/jaas.conf"; ./bin/zookeeper-
security-migration.sh --zookeeper.acl=secure --zookeeper.connect=<ZooKeeperURL>
exit
```

For example:

```
su - kafka
cd /opt/kafka
KAFKA_OPTS="-Djava.security.auth.login.config=./config/jaas.conf"; ./bin/zookeeper-
security-migration.sh --zookeeper.acl=secure --zookeeper.connect=zoo1.my-
domain.com:2181
exit
```

## 4.9. ENCRYPTION AND AUTHENTICATION

AMQ Streams supports encryption and authentication, which is configured as part of the listener configuration.

### 4.9.1. Listener configuration

Encryption and authentication in Kafka brokers is configured per listener. For more information about Kafka listener configuration, see [Section 4.2, “Listeners”](#).

Each listener in the Kafka broker is configured with its own security protocol. The configuration property **listener.security.protocol.map** defines which listener uses which security protocol. It maps each listener name to its security protocol. Supported security protocols are:

#### PLAINTEXT

Listener without any encryption or authentication.

#### SSL

Listener using TLS encryption and, optionally, authentication using TLS client certificates.

#### SASL\_PLAINTEXT

Listener without encryption but with SASL-based authentication.

#### SASL\_SSL

Listener with TLS-based encryption and SASL-based authentication.

Given the following **listeners** configuration:

```
listeners=INT1://:9092,INT2://:9093,REPLICATION://:9094
```

the **listener.security.protocol.map** might look like this:

```
listener.security.protocol.map=INT1:SASL_PLAINTEXT,INT2:SASL_SSL,REPLICATION:SSL
```

This would configure the listener **INT1** to use unencrypted connections with SASL authentication, the listener **INT2** to use encrypted connections with SASL authentication and the **REPLICATION** interface to use TLS encryption (possibly with TLS client authentication). The same security protocol can be used multiple times. The following example is also a valid configuration:

```
listener.security.protocol.map=INT1:SSL,INT2:SSL,REPLICATION:SSL
```

Such a configuration would use TLS encryption and TLS authentication for all interfaces. The following chapters will explain in more detail how to configure TLS and SASL.

### 4.9.2. TLS Encryption

Kafka supports TLS for encrypting communication with Kafka clients.

In order to use TLS encryption and server authentication, a keystore containing private and public keys has to be provided. This is usually done using a file in the Java Keystore (JKS) format. A path to this file is set in the **ssl.keystore.location** property. The **ssl.keystore.password** property should be used to set the password protecting the keystore. For example:

```
ssl.keystore.location=/path/to/keystore/server-1.jks
ssl.keystore.password=123456
```

In some cases, an additional password is used to protect the private key. Any such password can be set using the **ssl.key.password** property.

Kafka is able to use keys signed by certification authorities as well as self-signed keys. Using keys signed

by certification authorities should always be the preferred method. In order to allow clients to verify the identity of the Kafka broker they are connecting to, the certificate should always contain the advertised hostname(s) as its Common Name (CN) or in the Subject Alternative Names (SAN).

It is possible to use different SSL configurations for different listeners. All options starting with **ssl.** can be prefixed with **listener.name.<NameOfTheListener>**, where the name of the listener has to be always in lower case. This will override the default SSL configuration for that specific listener. The following example shows how to use different SSL configurations for different listeners:

```
listeners=INT1://:9092,INT2://:9093,REPLICATION://:9094
listener.security.protocol.map=INT1:SSL,INT2:SSL,REPLICATION:SSL

# Default configuration - will be used for listeners INT1 and INT2
ssl.keystore.location=/path/to/keystore/server-1.jks
ssl.keystore.password=123456

# Different configuration for listener REPLICATION
listener.name.replication.ssl.keystore.location=/path/to/keystore/server-1.jks
listener.name.replication.ssl.keystore.password=123456
```

### Additional TLS configuration options

In addition to the main TLS configuration options described above, Kafka supports many options for fine-tuning the TLS configuration. For example, to enable or disable TLS / SSL protocols or cipher suites:

#### ssl.cipher.suites

List of enabled cipher suites. Each cipher suite is a combination of authentication, encryption, MAC and key exchange algorithms used for the TLS connection. By default, all available cipher suites are enabled.

#### ssl.enabled.protocols

List of enabled TLS / SSL protocols. Defaults to **TLSv1.2,TLSv1.1,TLSv1**.

For a complete list of supported Kafka broker configuration options, see [Appendix A, Broker configuration parameters](#).

### 4.9.3. Enabling TLS encryption

This procedure describes how to enable encryption in Kafka brokers.

#### Prerequisites

- AMQ Streams is [installed](#) on all hosts which will be used as Kafka brokers.

#### Procedure

1. Generate TLS certificates for all Kafka brokers in your cluster. The certificates should have their advertised and bootstrap addresses in their Common Name or Subject Alternative Name.
2. Edit the **/opt/kafka/config/server.properties** Kafka configuration file on all cluster nodes for the following:
  - Change the **listener.security.protocol.map** field to specify the **SSL** protocol for the listener where you want to use TLS encryption.

- Set the **ssl.keystore.location** option to the path to the JKS keystore with the broker certificate.
- Set the **ssl.keystore.password** option to the password you used to protect the keystore. For example:

```
listeners=UNENCRYPTED://:9092,ENCRYPTED://:9093,REPLICATION://:9094
listener.security.protocol.map=UNENCRYPTED:PLAINTEXT,ENCRYPTED:SSL,REPLICATION:PLAINTEXT
ssl.keystore.location=/path/to/keystore/server-1.jks
ssl.keystore.password=123456
```

3. (Re)start the Kafka brokers

### Additional resources

- For more information about configuring AMQ Streams, see [Section 2.8, “Configuring AMQ Streams”](#).
- For more information about running a Kafka cluster, see [Section 4.5, “Running a multi-node Kafka cluster”](#).
- For more information about configuring TLS encryption in clients, see:
  - [Appendix D, \*Producer configuration parameters\*](#)
  - [Appendix C, \*Consumer configuration parameters\*](#)

## 4.9.4. Authentication

For authentication, you can use:

- TLS client authentication based on X.509 certificates on encrypted connections
- A supported Kafka SASL (Simple Authentication and Security Layer) mechanism
- [OAuth 2.0 token based authentication](#)

### 4.9.4.1. TLS client authentication

TLS client authentication can be used only on connections which are already using TLS encryption. To use TLS client authentication, a truststore with public keys can be provided to the broker. These keys can be used to authenticate clients connecting to the broker. The truststore should be provided in Java Keystore (JKS) format and should contain public keys of the certification authorities. All clients with public and private keys signed by one of the certification authorities included in the truststore will be authenticated. The location of the truststore is set using field **ssl.truststore.location**. In case the truststore is password protected, the password should be set in the **ssl.truststore.password** property. For example:

```
ssl.truststore.location=/path/to/keystore/server-1.jks
ssl.truststore.password=123456
```

Once the truststore is configured, TLS client authentication has to be enabled using the **ssl.client.auth** property. This property can be set to one of three different values:



**none**

TLS client authentication is switched off. (Default value)

**requested**

TLS client authentication is optional. Clients will be asked to authenticate using TLS client certificate but they can choose not to.

**required**

Clients are required to authenticate using TLS client certificate.

When a client authenticates using TLS client authentication, the authenticated principal name is the distinguished name from the authenticated client certificate. For example, a user with a certificate which has a distinguished name **CN=someuser** will be authenticated with the following principal **CN=someuser,OU=Unknown,O=Unknown,L=Unknown,ST=Unknown,C=Unknown**. When TLS client authentication is not used and SASL is disabled, the principal name will be **ANONYMOUS**.

**4.9.4.2. SASL authentication**

SASL authentication is configured using Java Authentication and Authorization Service (JAAS). JAAS is also used for authentication of connections between Kafka and ZooKeeper. JAAS uses its own configuration file. The recommended location for this file is **/opt/kafka/config/jaas.conf**. The file has to be readable by the **kafka** user. When running Kafka, the location of this file is specified using Java system property **java.security.auth.login.config**. This property has to be passed to Kafka when starting the broker nodes:

```
KAFKA_OPTS="-Djava.security.auth.login.config=/path/to/my/jaas.conf"; bin/kafka-server-start.sh
```

SASL authentication is supported both through plain unencrypted connections as well as through TLS connections. SASL can be enabled individually for each listener. To enable it, the security protocol in **listener.security.protocol.map** has to be either **SASL\_PLAINTEXT** or **SASL\_SSL**.

SASL authentication in Kafka supports several different mechanisms:

**PLAIN**

Implements authentication based on username and passwords. Usernames and passwords are stored locally in Kafka configuration.

**SCRAM-SHA-256 and SCRAM-SHA-512**

Implements authentication using Salted Challenge Response Authentication Mechanism (SCRAM). SCRAM credentials are stored centrally in ZooKeeper. SCRAM can be used in situations where ZooKeeper cluster nodes are running isolated in a private network.

**GSSAPI**

Implements authentication against a Kerberos server.

**WARNING**

The **PLAIN** mechanism sends the username and password over the network in an unencrypted format. It should be therefore only be used in combination with TLS encryption.

The SASL mechanisms are configured via the JAAS configuration file. Kafka uses the JAAS context named **KafkaServer**. After they are configured in JAAS, the SASL mechanisms have to be enabled in the Kafka configuration. This is done using the **sasl.enabled.mechanisms** property. This property contains a comma-separated list of enabled mechanisms:

```
sasl.enabled.mechanisms=PLAIN,SCRAM-SHA-256,SCRAM-SHA-512
```

In case the listener used for inter-broker communication is using SASL, the property **sasl.mechanism.inter.broker.protocol** has to be used to specify the SASL mechanism which it should use. For example:

```
sasl.mechanism.inter.broker.protocol=PLAIN
```

The username and password which will be used for the inter-broker communication has to be specified in the **KafkaServer** JAAS context using the field **username** and **password**.

## SASL PLAIN

To use the PLAIN mechanism, the usernames and password which are allowed to connect are specified directly in the JAAS context. The following example shows the context configured for SASL PLAIN authentication. The example configures three different users:

- **admin**
- **user1**
- **user2**

```
KafkaServer {  
  org.apache.kafka.common.security.plain.PlainLoginModule required  
  user_admin="123456"  
  user_user1="123456"  
  user_user2="123456";  
};
```

The JAAS configuration file with the user database should be kept in sync on all Kafka brokers.

When SASL PLAIN is also used for inter-broker authentication, the **username** and **password** properties should be included in the JAAS context:

```
KafkaServer {  
  org.apache.kafka.common.security.plain.PlainLoginModule required  
  username="admin"  
  password="123456"  
  user_admin="123456"  
  user_user1="123456"  
  user_user2="123456";  
};
```

## SASL SCRAM

SCRAM authentication in Kafka consists of two mechanisms: **SCRAM-SHA-256** and **SCRAM-SHA-512**. These mechanisms differ only in the hashing algorithm used - SHA-256 versus stronger SHA-512. To enable SCRAM authentication, the JAAS configuration file has to include the following configuration:

```
KafkaServer {
  org.apache.kafka.common.security.scram.ScramLoginModule required;
};
```

When enabling SASL authentication in the Kafka configuration file, both SCRAM mechanisms can be listed. However, only one of them can be chosen for the inter-broker communication. For example:

```
sasl.enabled.mechanisms=SCRAM-SHA-256,SCRAM-SHA-512
sasl.mechanism.inter.broker.protocol=SCRAM-SHA-512
```

User credentials for the SCRAM mechanism are stored in ZooKeeper. The **kafka-configs.sh** tool can be used to manage them. For example, run the following command to add user `user1` with password `123456`:

```
bin/kafka-configs.sh --zookeeper zoo1.my-domain.com:2181 --alter --add-config 'SCRAM-SHA-256=[password=123456],SCRAM-SHA-512=[password=123456]' --entity-type users --entity-name user1
```

To delete a user credential use:

```
bin/kafka-configs.sh --zookeeper zoo1.my-domain.com:2181 --alter --delete-config 'SCRAM-SHA-512' --entity-type users --entity-name user1
```

## SASL GSSAPI

The SASL mechanism used for authentication using Kerberos is called **GSSAPI**. To configure Kerberos SASL authentication, the following configuration should be added to the JAAS configuration file:

```
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";
};
```

The domain name in the Kerberos principal has to be always in upper case.

In addition to the JAAS configuration, the Kerberos service name needs to be specified in the **sasl.kerberos.service.name** property in the Kafka configuration:

```
sasl.enabled.mechanisms=GSSAPI
sasl.mechanism.inter.broker.protocol=GSSAPI
sasl.kerberos.service.name=kafka
```

## Multiple SASL mechanisms

Kafka can use multiple SASL mechanisms at the same time. The different JAAS configurations can be all added to the same context:

```
KafkaServer {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  user_admin="123456"
  user_user1="123456"
  user_user2="123456";
```

```

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.scram.ScramLoginModule required;
};

```

When multiple mechanisms are enabled, clients will be able to choose the mechanism which they want to use.

#### 4.9.5. Enabling TLS client authentication

This procedure describes how to enable TLS client authentication in Kafka brokers.

##### Prerequisites

- AMQ Streams is [installed](#) on all hosts which will be used as Kafka brokers.
- TLS encryption is [enabled](#).

##### Procedure

1. Prepare a JKS truststore containing the public key of the certification authority used to sign the user certificates.
2. Edit the `/opt/kafka/config/server.properties` Kafka configuration file on all cluster nodes for the following:
  - Set the **ssl.truststore.location** option to the path to the JKS truststore with the certification authority of the user certificates.
  - Set the **ssl.truststore.password** option to the password you used to protect the truststore.
  - Set the **ssl.client.auth** option to **required**.  
For example:

```

ssl.truststore.location=/path/to/truststore.jks
ssl.truststore.password=123456
ssl.client.auth=required

```

3. (Re)start the Kafka brokers

##### Additional resources

- For more information about configuring AMQ Streams, see [Section 2.8, "Configuring AMQ Streams"](#).
- For more information about running a Kafka cluster, see [Section 4.5, "Running a multi-node Kafka cluster"](#).
- For more information about configuring TLS encryption in clients, see:

- [Appendix D, Producer configuration parameters](#)
- [Appendix C, Consumer configuration parameters](#)

### 4.9.6. Enabling SASL PLAIN authentication

This procedure describes how to enable SASL PLAIN authentication in Kafka brokers.

#### Prerequisites

- AMQ Streams is [installed](#) on all hosts which will be used as Kafka brokers.

#### Procedure

1. Edit or create the `/opt/kafka/config/jaas.conf` JAAS configuration file. This file should contain all your users and their passwords. Make sure this file is the same on all Kafka brokers.

For example:

```
KafkaServer {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  user_admin="123456"
  user_user1="123456"
  user_user2="123456";
};
```

2. Edit the `/opt/kafka/config/server.properties` Kafka configuration file on all cluster nodes for the following:

- Change the `listener.security.protocol.map` field to specify the **SASL\_PLAINTEXT** or **SASL\_SSL** protocol for the listener where you want to use SASL PLAIN authentication.

- Set the `sasl.enabled.mechanisms` option to **PLAIN**.

For example:

```
listeners=INSECURE://:9092,AUTHENTICATED://:9093,REPLICATION://:9094
listener.security.protocol.map=INSECURE:PLAINTEXT,AUTHENTICATED:SASL_PLAINTEXT,REPLICATION:PLAINTEXT
sasl.enabled.mechanisms=PLAIN
```

3. (Re)start the Kafka brokers using the `KAFKA_OPTS` environment variable to pass the JAAS configuration to Kafka brokers.

```
su - kafka
export KAFKA_OPTS="-Djava.security.auth.login.config=/opt/kafka/config/jaas.conf";
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
```

#### Additional resources

- For more information about configuring AMQ Streams, see [Section 2.8, “Configuring AMQ Streams”](#).
- For more information about running a Kafka cluster, see [Section 4.5, “Running a multi-node Kafka cluster”](#).

- For more information about configuring SASL PLAIN authentication in clients, see:
  - [Appendix D, \*Producer configuration parameters\*](#)
  - [Appendix C, \*Consumer configuration parameters\*](#)

### 4.9.7. Enabling SASL SCRAM authentication

This procedure describes how to enable SASL SCRAM authentication in Kafka brokers.

#### Prerequisites

- AMQ Streams is [installed](#) on all hosts which will be used as Kafka brokers.

#### Procedure

1. Edit or create the `/opt/kafka/config/jaas.conf` JAAS configuration file. Enable the **ScramLoginModule** for the **KafkaServer** context. Make sure this file is the same on all Kafka brokers.

For example:

```
KafkaServer {
    org.apache.kafka.common.security.scram.ScramLoginModule required;
};
```

2. Edit the `/opt/kafka/config/server.properties` Kafka configuration file on all cluster nodes for the following:
  - Change the **listener.security.protocol.map** field to specify the **SASL\_PLAINTEXT** or **SASL\_SSL** protocol for the listener where you want to use SASL SCRAM authentication.
  - Set the **sasl.enabled.mechanisms** option to **SCRAM-SHA-256** or **SCRAM-SHA-512**.

For example:

```
listeners=INSECURE://:9092,AUTHENTICATED://:9093,REPLICATION://:9094
listener.security.protocol.map=INSECURE:PLAINTEXT,AUTHENTICATED:SASL_PLAINTEXT,REPLICATION:PLAINTEXT
sasl.enabled.mechanisms=SCRAM-SHA-512
```

3. (Re)start the Kafka brokers using the `KAFKA_OPTS` environment variable to pass the JAAS configuration to Kafka brokers.

```
su - kafka
export KAFKA_OPTS="-Djava.security.auth.login.config=/opt/kafka/config/jaas.conf";
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
```

#### Additional resources

- For more information about configuring AMQ Streams, see [Section 2.8, "Configuring AMQ Streams"](#).
- For more information about running a Kafka cluster, see [Section 4.5, "Running a multi-node Kafka cluster"](#).

- For more information about adding SASL SCRAM users, see [Section 4.9.8, “Adding SASL SCRAM users”](#).
- For more information about deleting SASL SCRAM users, see [Section 4.9.9, “Deleting SASL SCRAM users”](#).
- For more information about configuring SASL SCRAM authentication in clients, see:
  - [Appendix D, Producer configuration parameters](#)
  - [Appendix C, Consumer configuration parameters](#)

### 4.9.8. Adding SASL SCRAM users

This procedure describes how to add new users for authentication using SASL SCRAM.

#### Prerequisites

- AMQ Streams is [installed](#) on all hosts which will be used as Kafka brokers.
- SASL SCRAM authentication is [enabled](#).

#### Procedure

- Use the **kafka-configs.sh** tool to add new SASL SCRAM users.

```
bin/kafka-configs.sh --zookeeper <ZooKeeperAddress> --alter --add-config 'SCRAM-SHA-512=[password=<Password>]' --entity-type users --entity-name <Username>
```

For example:

```
bin/kafka-configs.sh --zookeeper zoo1.my-domain.com:2181 --alter --add-config 'SCRAM-SHA-512=[password=123456]' --entity-type users --entity-name user1
```

#### Additional resources

- For more information about configuring SASL SCRAM authentication in clients, see:
  - [Appendix D, Producer configuration parameters](#)
  - [Appendix C, Consumer configuration parameters](#)

### 4.9.9. Deleting SASL SCRAM users

This procedure describes how to remove users when using SASL SCRAM authentication.

#### Prerequisites

- AMQ Streams is [installed](#) on all hosts which will be used as Kafka brokers.
- SASL SCRAM authentication is [enabled](#).

#### Procedure

- Use the **kafka-configs.sh** tool to delete SASL SCRAM users.

```
bin/kafka-configs.sh --zookeeper <ZooKeeperAddress> --alter --delete-config 'SCRAM-SHA-512' --entity-type users --entity-name <Username>
```

For example:

```
bin/kafka-configs.sh --zookeeper zoo1.my-domain.com:2181 --alter --delete-config 'SCRAM-SHA-512' --entity-type users --entity-name user1
```

### Additional resources

- For more information about configuring SASL SCRAM authentication in clients, see:
  - [Appendix D, Producer configuration parameters](#)
  - [Appendix C, Consumer configuration parameters](#)

## 4.10. USING OAUTH 2.0 TOKEN-BASED AUTHENTICATION

AMQ Streams supports the use of OAuth 2.0 authentication using the *SASL OAUTHBEARER* mechanism.

OAuth 2.0 enables standardized token based authentication and authorization between applications, using a central authorization server to issue tokens that grant limited access to resources.

In AMQ Streams, OAuth 2.0 is supported for authentication with OAuth 2.0 compliant authorization servers. OAuth 2.0 token-based authorization is also supported when using Keycloak as the authorization server, making use of its Authorization Services functionality to centrally manage users' permissions to Kafka resources. However, OAuth 2.0 authentication can be used in conjunction with [ACL-based Kafka authorization](#) regardless of the authorization server used.

Using OAuth 2.0 token-based authentication, application clients can access resources on application servers (called *resource servers*) without exposing account credentials.

The application client passes an access token as a means of authenticating, which application servers can also use to determine the level of access to grant. The authorization server handles the granting of access and inquiries about access.

In the context of AMQ Streams:

- Kafka brokers act as OAuth 2.0 resource servers
- Kafka clients act as OAuth 2.0 application clients

Kafka clients authenticate to Kafka brokers. The brokers and clients communicate with the OAuth 2.0 authorization server, as necessary, to obtain or validate access tokens.

For a deployment of AMQ Streams, OAuth 2.0 integration provides:

- Server-side OAuth 2.0 support for Kafka brokers
- Client-side OAuth 2.0 support for Kafka Mirror Maker, Kafka Connect and the Kafka Bridge

### Additional resources



- [OAuth 2.0 site](#)

### 4.10.1. OAuth 2.0 authentication mechanism

The Kafka *SASL OAUTHBEARER* mechanism is used to establish authenticated sessions with a Kafka broker.

A Kafka client initiates a session with the Kafka broker using the *SASL OAUTHBEARER* mechanism for credentials exchange, where credentials take the form of an access token.

Kafka brokers and clients need to be configured to use OAuth 2.0.

#### 4.10.1.1. Configuring OAuth 2.0 with properties or variables

You can configure OAuth 2.0 settings using Java Authentication and Authorization Service (JAAS) properties or environment variables.

- JAAS properties are configured in the **server.properties** configuration file, and passed as key-values pairs of the **listener.name.<listener-name>.oauthbearer.sasl.jaas.config** property.
- Using environment variables, you still need the **listener.name.<listener-name>.oauthbearer.sasl.jaas.config** in the **server.properties** file, but you can omit the properties.  
You can use capitalized or upper-case environment variable naming conventions.

### 4.10.2. OAuth 2.0 Kafka broker configuration

Kafka broker configuration for OAuth 2.0 involves:

- Creating the OAuth 2.0 client in the authorization server
- Configuring OAuth 2.0 authentication in the Kafka cluster



#### NOTE

In relation to the authorization server, Kafka brokers and Kafka clients are both regarded as OAuth 2.0 clients.

#### 4.10.2.1. OAuth 2.0 client configuration on an authorization server

To configure a Kafka broker to validate the token received during session initiation, the recommended approach is to create a OAuth 2.0 *client* definition in an authorization server, configured as *confidential*, with the following client credentials enabled:

- Client ID of **kafka-broker** (for example)
- Client ID and secret as the authentication mechanism



#### NOTE

You only need to use a client ID and secret when using a non-public introspection endpoint of the authorization server. The credentials are not typically required when using public authorization server endpoints, as with fast local JWT token validation.

#### 4.10.2.2. OAuth 2.0 authentication configuration in the Kafka cluster

To use OAuth 2.0 authentication in the Kafka cluster, you enable a listener configuration for your Kafka cluster in the Kafka **server.properties** file. A minimum configuration is required. You can also configure a TLS listener, where TLS is used for inter-broker communication.

You can configure the broker for token validation by the authorization server using the:

- *JWKS* endpoint in combination with signed JWT-formatted access tokens
- *Introspection* endpoint

The minimum configuration shown here applies a *global* listener configuration. This means that inter-broker communication, goes through the same listener as application clients.

To enable OAuth 2.0 configuration for a specific listener, you specify **listener.name.client.sasl.enabled.mechanisms** instead of **sasl.enabled.mechanisms**, which is shown in the example TLS listener configuration.

#### Minimum listener configuration for OAuth 2.0 authentication using a JWKS endpoint

```
sasl.enabled.mechanisms=OAUTHBEARER 1
listeners=CLIENT://0.0.0.0:9092 2
listener.security.protocol.map=CLIENT:SASL_PLAINTEXT 3
listener.name.client.sasl.enabled.mechanisms=OAUTHBEARER 4
sasl.mechanism.inter.broker.protocol=OAUTHBEARER 5
inter.broker.listener.name=CLIENT 6
listener.name.client.oauthbearer.sasl.server.callback.handler.class=io.strimzi.kafka.oauth.server.JaasServerOauthValidatorCallbackHandler 7
listener.name.client.oauthbearer.sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required \ 8
  oauth.valid.issuer.uri="https://<auth-server-address>" \ 9
  oauth.jwks.endpoint.uri="https://<auth-server-address>/jwks" \ 10
  oauth.username.claim="preferred_username" \ 11
  oauth.client.id="kafka-broker" \ 12
  oauth.client.secret="kafka-secret" \ 13
  oauth.token.endpoint.uri="https://<auth-server-address>/token" ; 14
listener.name.client.oauthbearer.sasl.login.callback.handler.class=io.strimzi.kafka.oauth.client.JaasClientOauthLoginCallbackHandler 15
```

- 1 Enables the *OAUTHBEARER* as *SASL* mechanism for credentials exchange over *SASL*.
- 2 Configures a listener for client applications to connect. The system **hostname** is used as an advertised hostname, which clients must resolve in order to reconnect.
- 3 Specifies the channel protocol for the listener. **SASL\_SSL** is for TLS. **SASL\_PLAINTEXT** is used for an unencrypted connection (no TLS), but there is risk of eavesdropping and interception at the TCP connection layer.
- 4 Specifies *OAUTHBEARER* as *SASL* for client listeners.
- 5 Specifies *OAUTHBEARER* as *SASL* for inter-broker communication.
- 6 Specifies the listener for inter-broker communication. The specification is required for the configuration to be valid.

- 7 Configures OAuth 2.0 authentication on the client listener.
- 8 Configures authentication settings for client and inter-broker communication. The **oauth.client.id**, **oauth.client.secret**, and **auth.token.endpoint.uri** properties relate to inter-broker configuration.
- 9 A valid issuer URI. Only access tokens issued by this issuer will be accepted.
- 10 The JWKS endpoint URL. For example, *https://<auth-server-address>/auth/realms/master/protocol/openid-connect/certs*.
- 11 The token claim (or key) that contains the actual user name in the token. The user name is the *principal* used to identify the user. The value will depend on the authentication flow and the authorization server used.
- 12 Client ID of the Kafka broker, which is the same for all brokers. This is the [client registered with the authorization server as kafka-broker](#).
- 13 Secret for the Kafka broker, which is the same for all brokers.
- 14 The OAuth 2.0 token endpoint URL to your authorization server. For production, always use HTTPS.
- 15 Enables (and is only required for) OAuth 2.0 authentication for inter-broker communication.

### TLS listener configuration for OAuth 2.0 authentication

```

sasl.enabled.mechanisms=
listeners=REPLICATION://kafka:9091,CLIENT://kafka:9092 1
listener.security.protocol.map=REPLICATION:SSL,CLIENT:SASL_PLAINTEXT 2
listener.name.client.sasl.enabled.mechanisms=OAUTHBEARER
inter.broker.listener.name=REPLICATION
listener.name.replication.ssl.keystore.password=<my-keystore-password> 3
listener.name.replication.ssl.truststore.password=<my-truststore-password>
listener.name.replication.ssl.keystore.type=JKS
listener.name.replication.ssl.truststore.type=JKS
listener.name.replication.ssl.endpoint.identification.algorithm=HTTPS 4
listener.name.replication.ssl.secure.random.implementation=SHA1PRNG 5
listener.name.replication.ssl.keystore.location=<path-to-keystore> 6
listener.name.replication.ssl.truststore.location=<path-to-truststore> 7
listener.name.replication.ssl.client.auth=required 8
listener.name.client.oauthbearer.sasl.server.callback.handler.class=io.strimzi.kafka.oauth.server.JaasServerOAuthValidatorCallbackHandler
listener.name.client.oauthbearer.sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required \
  oauth.valid.issuer.uri="https://<auth-server-address>" \
  oauth.jwks.endpoint.uri="https://<auth-server-address>/jwks" \
  oauth.username.claim="preferred_username" ; 9

```

- 1 Separate configurations are required for inter-broker communication and client applications.
- 2 Configures the *REPLICATION* listener to use TLS, and the *CLIENT* listener to use SASL over an unencrypted channel. The client could use an encrypted channel (**SASL\_SSL**) in a production environment.

- 3 The **ssl** properties define the TLS configuration.
- 4 Random number generator implementation. If not set, the Java platform SDK default is used.
- 5 Hostname verification. If set to an empty string, the hostname verification is turned off. If not set, the default value is HTTPS, which enforces hostname verification for server certificates.
- 6 Path to the keystore for the listener.
- 7 Path to the truststore for the listener.
- 8 Specifies that clients of the *REPLICATION* listener have to authenticate with a client certificate when establishing a TLS connection (used for inter-broker connectivity).
- 9 Configures the *CLIENT* listener for OAuth 2.0. Connectivity with the authorization server should use secure HTTPS connections.

#### 4.10.2.3. Fast local JWT token validation configuration

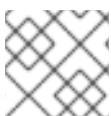
Fast local JWT token validation checks a JWT token signature locally.

The local check ensures that a token:

- Conforms to type by containing a (*typ*) claim value of **Bearer** for an access token
- Is valid (not expired)
- Has an issuer that matches a **validIssuerURI**

You specify a *valid issuer URI* when you configure the listener, so that any tokens not issued by the authorization server are rejected.

The authorization server does not need to be contacted during fast local JWT token validation. You activate fast local JWT token validation by specifying a *JWks endpoint URI* exposed by the OAuth 2.0 authorization server. The endpoint contains the public keys used to validate signed JWT tokens, which are sent as credentials by Kafka clients.



#### NOTE

All communication with the authorization server should be performed using HTTPS.

For a TLS listener, you can configure a certificate *truststore* and point to the truststore file.

#### Example properties file for fast local JWT token validation

```
listener.name.client.oauthbearer.sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required \
oauth.valid.issuer.uri="https://<auth-server-address>" \
oauth.jwks.endpoint.uri="https://<auth-server-address>/jwks" \
oauth.username.claim="preferred_username" \
oauth.ssl.truststore.location="<path-to-truststore-p12>" \
oauth.ssl.truststore.password="<my-password>" \
oauth.ssl.truststore.type="PKCS12" ;
```

Instead of specifying the configuration for the JWT token validation in the **server.properties** file, you can use environment variables.

### Example environment variables for fast local JWT token validation

```
export OAUTH_VALID_ISSUER_URI=https://<auth-server-address> 1
export OAUTH_JWKS_ENDPOINT_URI=https://<auth-server-address>/jwks 2
export OAUTH_JWKS_REFRESH_SECONDS=300 3
export OAUTH_JWKS_EXPIRY_SECONDS=360 4
export OAUTH_USERNAME_CLAIM=preferred_username 5
export OAUTH_SSL_TRUSTSTORE_LOCATION=<path-to-truststore-p12> 6
export OAUTH_SSL_TRUSTSTORE_PASSWORD=<my-password> 7
export OAUTH_SSL_TRUSTSTORE_TYPE=pkcs12 8
```

- 1 A valid issuer URI. Only access tokens issued by this issuer will be accepted.
- 2 The JWKS endpoint URL. For example, `https://<auth-server-address>/auth/realms/master/protocol/openid-connect/certs`.
- 3 The period between endpoint refreshes (default 300).
- 4 The duration the JWKS certificates are considered valid before they expire. Default is **360** seconds. If you specify a longer time, consider the risk of allowing access to revoked certificates.
- 5 The token claim (or key) that contains the actual user name in the token. The user name is the *principal* used to identify the user. The value will depend on the authentication flow and the authorization server used.
- 6 The location of the truststore used in the TLS configuration.
- 7 Password to access the truststore.
- 8 The truststore type in PKCS #12 format.

#### 4.10.2.4. OAuth 2.0 introspection endpoint configuration

Token validation using an OAuth 2.0 introspection endpoint treats a received access token as opaque. The Kafka broker sends an access token to the introspection endpoint, which responds with the token information necessary for validation. Importantly, it returns up-to-date information if the specific access token is valid, and also information about when the token expires.

To configure OAuth 2.0 introspection-based validation, you specify an *introspection endpoint URI* rather than the JWKS endpoint URI specified for fast local JWT token validation. Depending on the authorization server, you typically have to specify a *client ID* and *client secret*, because the introspection endpoint is usually protected.

### Example properties file for an introspection endpoint

```
listener.name.client.oauthbearer.sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required \
  oauth.introspection.endpoint.uri="https://<auth-server-address>/introspection" \
  oauth.client.id="kafka-broker" \
  oauth.client.secret="kafka-broker-secret" \
```

```

oauth.ssl.truststore.location="PATH_TO_P12_FILE" \
oauth.ssl.truststore.password="TRUSTSTORE_PASSWORD" \
oauth.ssl.truststore.type="PKCS12" ;

```

Instead of specifying the configuration for the endpoint in the **server.properties** file, you can use environment variables.

### Example environment variables for an introspection endpoint

```

export OAUTH_INTROSPECTION_ENDPOINT_URI=https://<auth-server-address>/introspection 1
export OAUTH_CLIENT_ID=kafka-broker 2
export OAUTH_CLIENT_SECRET=kafka-broker-secret 3
export OAUTH_SSL_TRUSTSTORE_LOCATION=<path-to-truststore-p12> 4
export OAUTH_SSL_TRUSTSTORE_PASSWORD=<my-password> 5
export OAUTH_SSL_TRUSTSTORE_TYPE=pkcs12 6

```

- 1 The OAuth 2.0 introspection endpoint URL. For example, *https://<auth-server-address>/auth/realms/master/protocol/openid-connect/token/introspect*.
- 2 Client ID of the Kafka broker.
- 3 Secret for the Kafka broker.
- 4 The location of the truststore used in the TLS configuration.
- 5 Password to access the truststore.
- 6 The truststore type in PKCS #12 format.

#### 4.10.3. OAuth 2.0 Kafka client configuration

A Kafka client is configured with either:

- The Credentials required to obtain a valid access token from an authorization server (client ID and Secret)
- A valid long-lived access token or refresh token, obtained using tools provided by an authorization server

Credentials are never sent to the Kafka broker. The only information ever sent to the Kafka broker is an access token. When a client obtains an access token, no further communication with the authorization server is needed.

The simplest mechanism is authentication with a client ID and Secret. Using a long-lived access token, or a long-lived refresh token, adds more complexity because there is additional dependency on authorization server tools.



#### NOTE

If you are using long-lived access tokens, you may need to configure the client in the authorization server to increase the maximum lifetime of the token.

If the Kafka client is not configured with an access token directly, the client exchanges credentials for an access token during Kafka session initiation by contacting the authorization server. The Kafka client exchanges either:

- Client ID and Secret
- Client ID, refresh token, and (optionally) a Secret

#### 4.10.4. OAuth 2.0 client authentication flow

In this section, we explain and visualize the communication flow between Kafka client, Kafka broker, and authorization server during Kafka session initiation. The flow depends on the client and server configuration.

When a Kafka client sends an access token as credentials to a Kafka broker, the token needs to be validated.

Depending on the authorization server used, and the configuration options available, you may prefer to use:

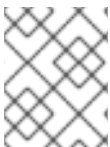
- Fast local token validation based on JWT signature checking and local token introspection, without contacting the authorization server
- An OAuth 2.0 introspection endpoint provided by the authorization server

Using fast local token validation requires the authorization server to provide a JWKS endpoint with public certificates that are used to validate signatures on the tokens.

Another option is to use an OAuth 2.0 introspection endpoint on the authorization server. Each time a new Kafka broker connection is established, the broker passes the access token received from the client to the authorization server, and checks the response to confirm whether or not the token is valid.

Kafka client credentials can also be configured for:

- Direct local access using a previously generated long-lived access token
- Contact with the authorization server for a new access token to be issued



#### NOTE

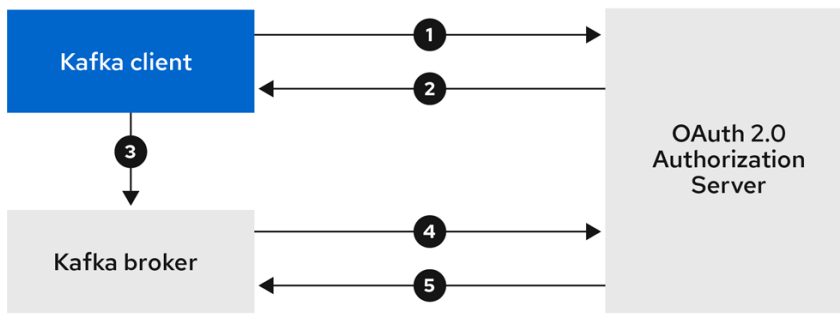
An authorization server might only allow the use of opaque access tokens, which means that local token validation is not possible.

#### 4.10.4.1. Example client authentication flows

Here you can see the communication flows, for different configurations of Kafka clients and brokers, during Kafka session authentication.

- [Client using client ID and secret, with broker delegating validation to authorization server](#)
- [Client using client ID and secret, with broker performing fast local token validation](#)
- [Client using long-lived access token, with broker delegating validation to authorization server](#)
- [Client using long-lived access token, with broker performing fast local validation](#)

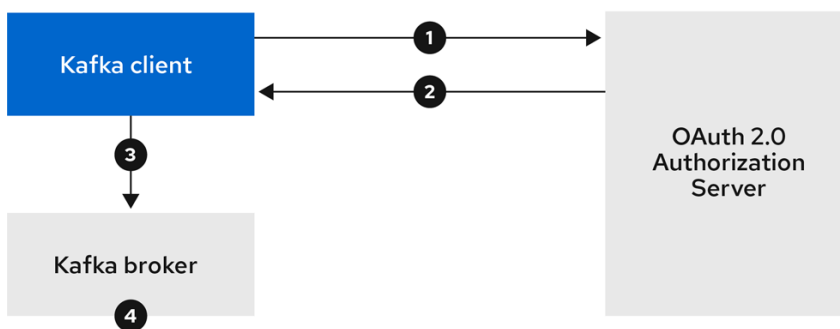
### Client using client ID and secret, with broker delegating validation to authorization server



AMQ\_46\_1019

1. Kafka client requests access token from authorization server, using client ID and secret, and optionally a refresh token.
2. Authorization server generates a new access token.
3. Kafka client authenticates with the Kafka broker using the *SASL OAUTHBEARER* mechanism to pass the access token.
4. Kafka broker validates the access token by calling a token introspection endpoint on authorization server, using its own client ID and secret.
5. Kafka client session is established if the token is valid.

### Client using client ID and secret, with broker performing fast local token validation

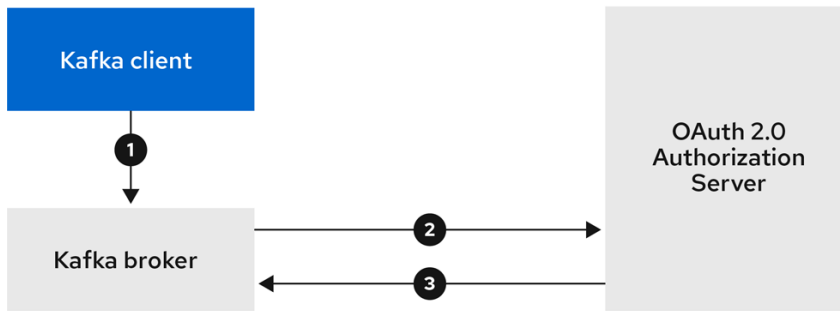


AMQ\_46\_1019

1. Kafka client authenticates with authorization server from the token endpoint, using a client ID and secret, and optionally a refresh token.
2. Authorization server generates a new access token.
3. Kafka client authenticates with the Kafka broker using the *SASL OAUTHBEARER* mechanism to pass the access token.
4. Kafka broker validates the access token locally using a JWT token signature check, and local token introspection.

### Client using long-lived access token, with broker delegating validation to authorization server





AMQ\_46\_1019

1. Kafka client authenticates with the Kafka broker using the *SASL OAUTHBEARER* mechanism to pass the long-lived access token.
2. Kafka broker validates the access token by calling a token introspection endpoint on authorization server, using its own client ID and secret.
3. Kafka client session is established if the token is valid.

### Client using long-lived access token, with broker performing fast local validation



AMQ\_46\_1019

1. Kafka client authenticates with the Kafka broker using the *SASL OAUTHBEARER* mechanism to pass the long-lived access token.
2. Kafka broker validates the access token locally using JWT token signature check, and local token introspection.



#### WARNING

Fast local JWT token signature validation is suitable only for short-lived tokens as there is no check with the authorization server if a token has been revoked. Token expiration is written into the token, but revocation can happen at any time, so cannot be accounted for without contacting the authorization server. Any issued token would be considered valid until it expires.

### 4.10.5. Configuring OAuth 2.0 authentication

OAuth 2.0 is used for interaction between Kafka clients and AMQ Streams components.

In order to use OAuth 2.0 for AMQ Streams, you must:

1. [Configure an OAuth 2.0 authorization server for the AMQ Streams cluster and Kafka clients](#)
2. [Deploy or update the Kafka cluster with Kafka broker listeners configured to use OAuth 2.0](#)
3. [Update your Java-based Kafka clients to use OAuth 2.0](#)

#### 4.10.5.1. Configuring Red Hat Single Sign-On as an OAuth 2.0 authorization server

This procedure describes how to deploy Red Hat Single Sign-On as an authorization server and configure it for integration with AMQ Streams.

The authorization server provides a central point for authentication and authorization, and management of users, clients, and permissions. Red Hat Single Sign-On has a concept of realms where a *realm* represents a separate set of users, clients, permissions, and other configuration. You can use a default *master realm*, or create a new one. Each realm exposes its own OAuth 2.0 endpoints, which means that application clients and application servers all need to use the same realm.

To use OAuth 2.0 with AMQ Streams, you need a deployment of an authorization server to be able to create and manage authentication realms.



#### NOTE

If you already have Red Hat Single Sign-On deployed, you can skip the deployment step and use your current deployment.

#### Before you begin

You will need to be familiar with using Red Hat Single Sign-On.

For installation and administration instructions, see:

- [Server Installation and Configuration Guide](#)
- [Server Administration Guide](#)

#### Prerequisites

- AMQ Streams and Kafka are running

For the Red Hat Single Sign-On deployment:

- Check the [Red Hat Single Sign-On Supported Configurations](#)

#### Procedure

1. Install Red Hat Single Sign-On.  
You can install from a ZIP file or by using an RPM.
2. Log in to the Red Hat Single Sign-On Admin Console to create the OAuth 2.0 policies for AMQ Streams.  
Login details are provided when you deploy Red Hat Single Sign-On.
3. Create and enable a realm.  
You can use an existing master realm.
4. Adjust the session and token timeouts for the realm, if required.

5. Create a client called **kafka-broker**.
6. From the **Settings** tab, set:
  - **Access Type** to **Confidential**
  - **Standard Flow Enabled** to **OFF** to disable web login for this client
  - **Service Accounts Enabled** to **ON** to allow this client to authenticate in its own name
7. Click **Save** before continuing.
8. From the **Credentials** tab, take a note of the secret for using in your AMQ Streams Kafka cluster configuration.
9. Repeat the client creation steps for any application client that will connect to your Kafka brokers.  
Create a definition for each new client.

You will use the names as client IDs in your configuration.

## What to do next

After deploying and configuring the authorization server, [configure the Kafka brokers to use OAuth 2.0](#) .

### 4.10.5.2. Configuring OAuth 2.0 support for Kafka brokers

This procedure describes how to configure Kafka brokers so that the broker listeners are enabled to use OAuth 2.0 authentication using an authorization server.

OAuth 2.0 requires configuration of TLS listeners. Plain listeners are not recommended.

## Before you start

For more information on the configuration and authentication of Kafka broker listeners, see:

- [Listeners](#)
- [Encryption and authentication](#)

For a description of the properties used in the listener configuration, see:

- [OAuth 2.0 Kafka broker configuration](#)

## Prerequisites

- AMQ Streams and Kafka are running
- An OAuth 2.0 authorization server is deployed

## Procedure

1. Configure the Kafka broker listener configuration in the **server.properties** file.  
For example:

```
sasl.enabled.mechanisms=OAUTHBEARER
listeners=CLIENT://0.0.0.0:9092
```

```

listener.security.protocol.map=CLIENT:SASL_PLAINTEXT
listener.name.client.sasl.enabled.mechanisms=OAUTHBEARER
sasl.mechanism.inter.broker.protocol=OAUTHBEARER
inter.broker.listener.name=CLIENT
listener.name.client.oauthbearer.sasl.server.callback.handler.class=io.strimzi.kafka.oauth.server.JaasServerOauthValidatorCallbackHandler
listener.name.client.oauthbearer.sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required ;
listener.name.client.oauthbearer.sasl.login.callback.handler.class=io.strimzi.kafka.oauth.client.JaasClientOauthLoginCallbackHandler

```

2. Add listener environment variables to configure broker connection options. The examples here show additional configuration options.

### Example 1: Local token validation using a JWKS endpoint configuration

```

export OAUTH_VALID_ISSUER_URI=https://<auth-server-address>
export OAUTH_JWKS_ENDPOINT_URI=https://<auth-server-address>/jwks
export OAUTH_JWKS_REFRESH_SECONDS=300
export OAUTH_JWKS_EXPIRY_SECONDS=360
export OAUTH_USERNAME_CLAIM=preferred_username
export OAUTH_SSL_TRUSTSTORE_LOCATION=<path-to-truststore-p12>
export OAUTH_SSL_TRUSTSTORE_PASSWORD=<my-password>
export OAUTH_SSL_TRUSTSTORE_TYPE=pkcs12

```

### Example 2: Delegating token validation to the authorization server through the OAuth 2.0 introspection endpoint

```

export OAUTH_INTROSPECTION_ENDPOINT_URI=https://<auth-server-address>/introspection

```

3. If required, configure access to the authorization server. This step is normally required for a production environment, unless a technology like *service mesh* is used to configure secure channels outside containers.
  - a. Provide a custom truststore for connecting to a secured authorization server. SSL is always required for access to the authorization server. Set environment variables to configure the truststore.

For example:

```

export OAUTH_CLIENT_ID=kafka-broker
export OAUTH_CLIENT_SECRET=kafka-broker-secret
export OAUTH_SSL_TRUSTSTORE_LOCATION=<path-to-truststore-p12>
export OAUTH_SSL_TRUSTSTORE_PASSWORD=<my-password>
export OAUTH_SSL_TRUSTSTORE_TYPE=pkcs12

```

- b. If the certificate hostname does not match the access URL hostname, you can turn off certificate hostname validation:

```

export OAUTH_SSL_ENDPOINT_IDENTIFICATION_ALGORITHM=""

```

The check ensures that client connection to the authorization server is authentic. You may wish to turn off the validation in a non-production environment.

4. Configure the following environment variables according to your chosen authentication flow.

```
export OAUTH_TOKEN_ENDPOINT_URI=https://<auth-server-address>/token 1
export OAUTH_VALID_ISSUER_URI=https://<auth-server-address> 2
export OAUTH_CLIENT_ID=kafka-broker 3
export OAUTH_CLIENT_SECRET=kafka-broker-secret 4

export OAUTH_REFRESH_TOKEN=<my-refresh-token-for-kafka-brokers> 5
export OAUTH_ACCESS_TOKEN=<my-access-token-for-kafka-brokers> 6
```

- 1 The OAuth 2.0 token endpoint URL to your authorization server. For production, always use HTTPs. Required when **KeycloakRBACAuthorizer** is used, or an OAuth 2.0 enabled listener is used for inter-broker communication.
- 2 A valid issuer URI. Only access tokens issued by this issuer will be accepted. (Always required.)
- 3 The configured client ID of the Kafka broker, which is the same for all brokers. This is the [client registered with the authorization server as kafka-broker](#). Required when an introspection endpoint is used for token validation, or when **KeycloakRBACAuthorizer** is used.
- 4 The configured secret for the Kafka broker, which is the same for all brokers. When the broker must authenticate to the authorization server, either a client secret, access token or a refresh token has to be specified.
- 5 (Optional) A long-lived refresh token for Kafka brokers.
- 6 (Optional) A long-lived access token for Kafka brokers.

### What to do next

- [Configure your Kafka clients to use OAuth 2.0](#)

#### 4.10.5.3. Configuring Kafka Java clients to use OAuth 2.0

This procedure describes how to configure Kafka producer and consumer APIs to use OAuth 2.0 for interaction with Kafka brokers.

Add a client callback plugin to your *pom.xml* file, and configure the system properties.

### Prerequisites

- AMQ Streams and Kafka are running
- An OAuth 2.0 authorization server is deployed and configured for OAuth access to Kafka brokers
- Kafka brokers are configured for OAuth 2.0

### Procedure

1. Add the client library with OAuth 2.0 support to the **pom.xml** file for the Kafka client:

```
<dependency>
<groupId>io.strimzi</groupId>
<artifactId>kafka-oauth-client</artifactId>
<version>0.3.0.redhat-00001</version>
</dependency>
```

2. Configure the system properties for the callback:

For example:

```
System.setProperty(ClientConfig.OAUTH_TOKEN_ENDPOINT_URI, "https://<auth-server-address>/auth/realms/master/protocol/openid-connect/token"); 1
System.setProperty(ClientConfig.OAUTH_CLIENT_ID, "<client-name>"); 2
System.setProperty(ClientConfig.OAUTH_CLIENT_SECRET, "<client-secret>"); 3
```

- 1** URI of the authorization server token endpoint.
- 2** Client ID, which is the name used when creating the *client* in the authorization server.
- 3** Client secret created when creating the *client* in the authorization server.

3. Enable the *SASL OAUTHBEARER* mechanism on a TLS encrypted connection in the Kafka client configuration:

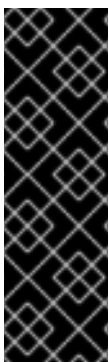
For example:

```
props.put("sasl.jaas.config",
"org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required;");
props.put("security.protocol", "SASL_SSL"); 1
props.put("sasl.mechanism", "OAUTHBEARER");
props.put("sasl.login.callback.handler.class",
"io.strimzi.kafka.oauth.client.JaasClientOAuthLoginCallbackHandler");
```

- 1** Here we use **SASL\_SSL** for use over TLS connections. Use **SASL\_PLAINTEXT** over unencrypted connections.

4. Verify that the Kafka client can access the Kafka brokers.

## 4.11. USING OAUTH 2.0 TOKEN-BASED AUTHORIZATION



### IMPORTANT

OAuth 2.0 authorization is a Technology Preview only. Technology Preview features are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend implementing any Technology Preview features in production environments. This Technology Preview feature provides early access to upcoming product innovations, enabling you to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

### Trying this feature

Red Hat Single Sign-On 7.3 does not support this Technology Preview of OAuth 2.0 token-based authorization. If you wish to try this feature, it is tested for use in a development environment with Keycloak 8.0.2 as the authorization server.

### Authorizing access to Kafka brokers

If you are using OAuth 2.0 with Keycloak for token-based authentication, you can also use Keycloak to configure authorization rules to constrain client access to Kafka brokers. Authentication establishes the identity of a user. Authorization decides the level of access for that user.

AMQ Streams supports the use of OAuth 2.0 token-based authorization through Keycloak [Authorization Services](#), which allows you to manage security policies and permissions centrally.

Security policies and permissions defined in Keycloak are used to grant access to resources on Kafka brokers. Users and clients are matched against policies that permit access to perform specific actions on Kafka brokers.

Kafka allows all users full access to brokers by default, and also provides the **SimpleACLAuthorizer** plugin to configure authorization based on Access Control Lists (ACLs). ZooKeeper stores ACL rules that grant or deny access to resources based on *username*. However, OAuth 2.0 token-based authorization with Keycloak offers far greater flexibility on how you wish to implement access control to Kafka brokers. In addition, you can configure your Kafka brokers to use OAuth 2.0 authorization and ACLs.

### Additional resources

- [Using OAuth 2.0 token based authentication](#)
- [ACL authorization](#)
- [Keycloak documentation](#)

#### 4.11.1. OAuth 2.0 authorization mechanism

OAuth 2.0 authorization in AMQ Streams uses Keycloak server Authorization Services REST endpoints to extend token-based authentication with Keycloak by applying defined security policies on a particular user, and providing a list of permissions granted on different resources for that user. Policies use roles and groups to match permissions to users. OAuth 2.0 authorization enforces permissions locally based on the received list of grants for the user from Keycloak Authorization Services.

##### 4.11.1.1. Kafka broker custom authorizer

A Keycloak *authorizer* (**KeycloakRBACAuthorizer**) is provided with AMQ Streams. To be able to use the Keycloak REST endpoints for Authorization Services provided by Keycloak, you configure a custom authorizer on the Kafka broker.

The authorizer fetches a list of granted permissions from the authorization server as needed, and enforces authorization locally on the Kafka Broker, making rapid authorization decisions for each client request.

#### 4.11.2. Configuring OAuth 2.0 authorization support

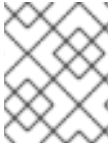
This procedure describes how to configure Kafka brokers to use OAuth 2.0 authorization using Keycloak Authorization Services.

### Before you begin

Consider the access you require or want to limit for certain users. You can use a combination of Keycloak *groups*, *roles*, *clients*, and *users* to configure access in Keycloak.

Typically, groups are used to match users based on organizational departments or geographical locations. And roles are used to match users based on their function.

With Keycloak, you can store users and groups in LDAP, whereas clients and roles cannot be stored this way. Storage and access to user data may be a factor in how you choose to configure authorization policies.



## NOTE

[Super users](#) always have unconstrained access to a Kafka broker regardless of the authorization implemented on the Kafka broker.

## Prerequisites

- AMQ Streams must be configured to use OAuth 2.0 with Keycloak for [token-based authentication](#). You use the same Keycloak server endpoint when you set up authorization.
- You need to understand how to manage policies and permissions for Keycloak Authorization Services, as described in the [Keycloak documentation](#).

## Procedure

1. Access the Keycloak Admin Console or use the Keycloak Admin CLI to enable Authorization Services for the Kafka broker client you created when setting up OAuth 2.0 authentication.
2. Use Authorization Services to define resources, authorization scopes, policies, and permissions for the client.
3. Bind the permissions to users and clients by assigning them roles and groups.
4. Configure the Kafka brokers to use Keycloak authorization.  
Add the following to the Kafka **server.properties** configuration file to install the authorizer in Kafka:

```
authorizer.class.name=io.strimzi.kafka.oauth.server.authorizer.KeycloakRBACAuthorizer  
principal.builder.class=io.strimzi.kafka.oauth.server.authorizer.JwtKafkaPrincipalBuilder
```

5. Add configuration for the Kafka brokers to access the authorization server and Authorization Services.

Here we show example configuration added as additional properties to **server.properties**, but you can also define them as environment variables using capitalized or upper-case naming conventions.

```
strimzi.authz.token.endpoint.uri="<https://<auth-server-  
address>/auth/realms/external/protocol/openid-connect/token" 1  
strimzi.authz.client.id="kafka" 2
```

- 1** The OAuth 2.0 token endpoint URL to Keycloak. For production, always use HTTPS.
- 2** The client ID of the OAuth 2.0 client definition in Keycloak that has Authorization Services enabled. Typically, **kafka** is used as the ID.



6. (Optional) Add configuration for specific Kafka clusters.

For example:

```
strimzi.authz.kafka.cluster.name="kafka-cluster" 1
```

- 1 The name of a specific Kafka cluster. Names are used to target permissions, making it possible to manage multiple clusters within the same Keycloak realm. The default value is **kafka-cluster**.

7. (Optional) Delegate to simple authorization.

For example:

```
strimzi.authz.delegate.to.kafka.acl="false" 1
```

- 1 Delegate authorization to Kafka **SimpleACLAuthorizer** if access is denied by Keycloak Authorization Services policies. The default is **false**.

8. (Optional) Add configuration for TLS connection to the authorization server.

For example:

```
strimzi.authz.ssl.truststore.location=<path-to-truststore> 1
strimzi.authz.ssl.truststore.password=<my-truststore-password> 2
strimzi.authz.ssl.truststore.type=JKS 3
strimzi.authz.ssl.secure.random.implementation=SHA1PRNG 4
strimzi.authz.ssl.endpoint.identification.algorithm=HTTPS 5
```

- 1 The path to the truststore that contain the certificates.
- 2 The password for the truststore.
- 3 The truststore type. If not set, the default Java keystore type is used.
- 4 Random number generator implementation. If not set, the Java platform SDK default is used.
- 5 Hostname verification. If set to an empty string, the hostname verification is turned off. If not set, the default value is **HTTPS**, which enforces hostname verification for server certificates.

9. Verify the configured permissions by accessing Kafka brokers as clients or users with specific roles, making sure they have the necessary access, or do not have the access they are not supposed to have.

## 4.12. LOGGING

Kafka brokers use Log4j as their logging infrastructure. Logging configuration is by default read from the **log4j.properties** configuration file which should be placed either in the `/opt/kafka/config/` directory or on the classpath. The location and name of the configuration file can be changed using the Java property **log4j.configuration** which can be passed to Kafka using the **KAFKA\_LOG4J\_OPTS** environment variable:

■

```
su - kafka
export KAFKA_LOG4J_OPTS="-Dlog4j.configuration=file:/my/path/to/log4j.config";
/opt/kafka/bin/kafka-server-start.sh /opt/kafka/config/server.properties
```

For more information about Log4j configurations, see [Log4j manual](#).

## CHAPTER 5. TOPICS

Messages in Kafka are always sent to or received from a topic. This chapter describes how to configure and manage Kafka topics.

### 5.1. PARTITIONS AND REPLICAS

Messages in Kafka are always sent to or received from a topic. A topic is always split into one or more partitions. Partitions act as shards. That means that every message sent by a producer is always written only into a single partition. Thanks to the sharding of messages into different partitions, topics are easy to scale horizontally.

Each partition can have one or more replicas, which will be stored on different brokers in the cluster. When creating a topic you can configure the number of replicas using the *replication factor*. *Replication factor* defines the number of copies which will be held within the cluster. One of the replicas for given partition will be elected as a leader. The leader replica will be used by the producers to send new messages and by the consumers to consume messages. The other replicas will be follower replicas. The followers replicate the leader.

If the leader fails, one of the followers will automatically become the new leader. Each server acts as a leader for some of its partitions and a follower for others so the load is well balanced within the cluster.



#### NOTE

The replication factor determines the number of replicas including the leader and the followers. For example, if you set the replication factor to **3**, then there will one leader and two follower replicas.

### 5.2. MESSAGE RETENTION

The message retention policy defines how long the messages will be stored on the Kafka brokers. It can be defined based on time, partition size or both.

For example, you can define that the messages should be kept:

- For 7 days
- Until the partition has 1GB of messages. Once the limit is reached, the oldest messages will be removed.
- For 7 days or until the 1GB limit has been reached. Whatever limit comes first will be used.



#### WARNING

Kafka brokers store messages in log segments. The messages which are past their retention policy will be deleted only when a new log segment is created. New log segments are created when the previous log segment exceeds the configured log segment size. Additionally, users can request new segments to be created periodically.

Additionally, Kafka brokers support a compacting policy.

For a topic with the compacted policy, the broker will always keep only the last message for each key. The older messages with the same key will be removed from the partition. Because compacting is a periodically executed action, it does not happen immediately when the new message with the same key are sent to the partition. Instead it might take some time until the older messages are removed.

For more information about the message retention configuration options, see [Section 5.5, “Topic configuration”](#).

## 5.3. TOPIC AUTO-CREATION

When a producer or consumer tries to send messages to or receive messages from a topic that does not exist, Kafka will, by default, automatically create that topic. This behavior is controlled by the **auto.create.topics.enable** configuration property which is set to **true** by default.

To disable it, set **auto.create.topics.enable** to **false** in the Kafka broker configuration file:

```
auto.create.topics.enable=false
```

## 5.4. TOPIC DELETION

Kafka offers the possibility to disable deletion of topics. This is configured through the **delete.topic.enable** property, which is set to **true** by default (that is, deleting topics is possible). When this property is set to **false** it will be not possible to delete topics and all attempts to delete topic will return success but the topic will not be deleted.

```
delete.topic.enable=false
```

## 5.5. TOPIC CONFIGURATION

Auto-created topics will use the default topic configuration which can be specified in the broker properties file. However, when creating topics manually, their configuration can be specified at creation time. It is also possible to change a topic’s configuration after it has been created. The main topic configuration options for manually created topics are:

### **cleanup.policy**

Configures the retention policy to **delete** or **compact**. The **delete** policy will delete old records. The **compact** policy will enable log compaction. The default value is **delete**. For more information about log compaction, see [Kafka website](#).

### **compression.type**

Specifies the compression which is used for stored messages. Valid values are **gzip**, **snappy**, **lz4**, **uncompressed** (no compression) and **producer** (retain the compression codec used by the producer). The default value is **producer**.

### **max.message.bytes**

The maximum size of a batch of messages allowed by the Kafka broker, in bytes. The default value is **100012**.

### **min.insync.replicas**

The minimum number of replicas which must be in sync for a write to be considered successful. The default value is **1**.

### **retention.ms**

Maximum number of milliseconds for which log segments will be retained. Log segments older than this value will be deleted. The default value is **604800000** (7 days).

#### **retention.bytes**

The maximum number of bytes a partition will retain. Once the partition size grows over this limit, the oldest log segments will be deleted. Value of **-1** indicates no limit. The default value is **-1**.

#### **segment.bytes**

The maximum file size of a single commit log segment file in bytes. When the segment reaches its size, a new segment will be started. The default value is **1073741824** bytes (1 gibibyte).

For list of all supported topic configuration options, see [Appendix B, Topic configuration parameters](#).

The defaults for auto-created topics can be specified in the Kafka broker configuration using similar options:

#### **log.cleanup.policy**

See **cleanup.policy** above.

#### **compression.type**

See **compression.type** above.

#### **message.max.bytes**

See **max.message.bytes** above.

#### **min.insync.replicas**

See **min.insync.replicas** above.

#### **log.retention.ms**

See **retention.ms** above.

#### **log.retention.bytes**

See **retention.bytes** above.

#### **log.segment.bytes**

See **segment.bytes** above.

#### **default.replication.factor**

Default replication factor for automatically created topics. Default value is **1**.

#### **num.partitions**

Default number of partitions for automatically created topics. Default value is **1**.

For list of all supported Kafka broker configuration options, see [Appendix A, Broker configuration parameters](#).

## 5.6. INTERNAL TOPICS

Internal topics are created and used internally by the Kafka brokers and clients. Kafka has several internal topics. These are used to store consumer offsets (**\_\_consumer\_offsets**) or transaction state (**\_\_transaction\_state**). These topics can be configured using dedicated Kafka broker configuration options starting with prefix **offsets.topic.** and **transaction.state.log.**. The most important configuration options are:

#### **offsets.topic.replication.factor**

Number of replicas for **\_\_consumer\_offsets** topic. The default value is **3**.

#### **offsets.topic.num.partitions**

Number of partitions for `__consumer_offsets` topic. The default value is **50**.

#### **transaction.state.log.replication.factor**

Number of replicas for `__transaction_state` topic. The default value is **3**.

#### **transaction.state.log.num.partitions**

Number of partitions for `__transaction_state` topic. The default value is **50**.

#### **transaction.state.log.min.isr**

Minimum number of replicas that must acknowledge a write to `__transaction_state` topic to be considered successful. If this minimum cannot be met, then the producer will fail with an exception. The default value is **2**.

## 5.7. CREATING A TOPIC

The **kafka-topics.sh** tool can be used to manage topics. **kafka-topics.sh** is part of the AMQ Streams distribution and can be found in the **bin** directory.

### Prerequisites

- AMQ Streams cluster is installed and running

### Creating a topic

1. Create a topic using the **kafka-topics.sh** utility and specify the following: ZooKeeper URL in the **--zookeeper** option. The new topic to be created in the **--create** option. Topic name in the **--topic** option. The number of partitions in the **--partitions** option. Replication factor in the **--replication-factor** option.

You can also override some of the default topic configuration options using the option **--config**. This option can be used multiple times to override different options.

```
bin/kafka-topics.sh --zookeeper <ZooKeeperAddress> --create --topic <TopicName> --
partitions <NumberOfPartitions> --replication-factor <ReplicationFactor> --config
<Option1>=<Value1> --config <Option2>=<Value2>
```

#### Example of the command to create a topic named **mytopic**

```
bin/kafka-topics.sh --zookeeper zoo1.my-domain.com:2181 --create --topic mytopic --
partitions 50 --replication-factor 3 --config cleanup.policy=compact --config
min.insync.replicas=2
```

2. Verify that the topic exists using **kafka-topics.sh**.

```
bin/kafka-topics.sh --zookeeper <ZooKeeperAddress> --describe --topic <TopicName>
```

#### Example of the command to describe a topic named **mytopic**

```
bin/kafka-topics.sh --zookeeper zoo1.my-domain.com:2181 --describe --topic mytopic
```

### Additional resources

- For more information about topic configuration, see [Section 5.5, “Topic configuration”](#).

- For list of all supported topic configuration options, see [Appendix B, Topic configuration parameters](#).

## 5.8. LISTING AND DESCRIBING TOPICS

The **kafka-topics.sh** tool can be used to list and describe topics. **kafka-topics.sh** is part of the AMQ Streams distribution and can be found in the **bin** directory.

### Prerequisites

- AMQ Streams cluster is installed and running
- Topic **mytopic** exists

### Describing a topic

1. Describe a topic using the **kafka-topics.sh** utility.
  - Specify the ZooKeeper URL in the **--zookeeper** option.
  - Use **--describe** option to specify that you want to describe a topic.
  - Topic name has to be specified in the **--topic** option.
  - When the **--topic** option is omitted, it will describe all available topics.

```
bin/kafka-topics.sh --zookeeper <ZooKeeperAddress> --describe --topic <TopicName>
```

### Example of the command to describe a topic named **mytopic**

```
bin/kafka-topics.sh --zookeeper zoo1.my-domain.com:2181 --describe --topic mytopic
```

The describe command will list all partitions and replicas which belong to this topic. It will also list all topic configuration options.

### Additional resources

- For more information about topic configuration, see [Section 5.5, "Topic configuration"](#).
- For more information about creating topics, see [Section 5.7, "Creating a topic"](#).

## 5.9. MODIFYING A TOPIC CONFIGURATION

The **kafka-configs.sh** tool can be used to modify topic configurations. **kafka-configs.sh** is part of the AMQ Streams distribution and can be found in the **bin** directory.

### Prerequisites

- AMQ Streams cluster is installed and running
- Topic **mytopic** exists

### Modify topic configuration

1. Use the **kafka-configs.sh** tool to get the current configuration.
  - Specify the ZooKeeper URL in the **--zookeeper** option.
  - Set the **--entity-type** as **topic** and **--entity-name** to the name of your topic.
  - Use **--describe** option to get the current configuration.

```
bin/kafka-configs.sh --zookeeper <ZooKeeperAddress> --entity-type topics --entity-name <TopicName> --describe
```

#### Example of the command to get configuration of a topic named **mytopic**

```
bin/kafka-configs.sh --zookeeper zoo1.my-domain.com:2181 --entity-type topics --entity-name mytopic --describe
```

2. Use the **kafka-configs.sh** tool to change the configuration.
  - Specify the ZooKeeper URL in the **--zookeeper** options.
  - Set the **--entity-type** as **topic** and **--entity-name** to the name of your topic.
  - Use **--alter** option to modify the current configuration.
  - Specify the options you want to add or change in the option **--add-config**.

```
bin/kafka-configs.sh --zookeeper <ZooKeeperAddress> --entity-type topics --entity-name <TopicName> --alter --add-config <Option>=<Value>
```

#### Example of the command to change configuration of a topic named **mytopic**

```
bin/kafka-configs.sh --zookeeper zoo1.my-domain.com:2181 --entity-type topics --entity-name mytopic --alter --add-config min.insync.replicas=1
```

3. Use the **kafka-configs.sh** tool to delete an existing configuration option.
  - Specify the ZooKeeper URL in the **--zookeeper** options.
  - Set the **--entity-type** as **topic** and **--entity-name** to the name of your topic.
  - Use **--delete-config** option to remove existing configuration option.
  - Specify the options you want to remove in the option **--remove-config**.

```
bin/kafka-configs.sh --zookeeper <ZooKeeperAddress> --entity-type topics --entity-name <TopicName> --alter --delete-config <Option>
```

#### Example of the command to change configuration of a topic named **mytopic**

```
bin/kafka-configs.sh --zookeeper zoo1.my-domain.com:2181 --entity-type topics --entity-name mytopic --alter --delete-config min.insync.replicas
```

#### Additional resources



- For more information about topic configuration, see [Section 5.5, “Topic configuration”](#).
- For more information about creating topics, see [Section 5.7, “Creating a topic”](#).
- For list of all supported topic configuration options, see [Appendix B, Topic configuration parameters](#).

## 5.10. DELETING A TOPIC

The **kafka-topics.sh** tool can be used to manage topics. **kafka-topics.sh** is part of the AMQ Streams distribution and can be found in the **bin** directory.

### Prerequisites

- AMQ Streams cluster is installed and running
- Topic **mytopic** exists

### Deleting a topic

1. Delete a topic using the **kafka-topics.sh** utility.
  - Specify the ZooKeeper URL in the **--zookeeper** option.
  - Use **--delete** option to specify that an existing topic should be deleted.
  - Topic name has to be specified in the **--topic** option.

```
bin/kafka-topics.sh --zookeeper <ZooKeeperAddress> --delete --topic <TopicName>
```

#### Example of the command to create a topic named **mytopic**

```
bin/kafka-topics.sh --zookeeper zoo1.my-domain.com:2181 --delete --topic mytopic
```

2. Verify that the topic was deleted using **kafka-topics.sh**.

```
bin/kafka-topics.sh --zookeeper <ZooKeeperAddress> --list
```

#### Example of the command to list all topics

```
bin/kafka-topics.sh --zookeeper zoo1.my-domain.com:2181 --list
```

### Additional resources

- For more information about creating topics, see [Section 5.7, “Creating a topic”](#).

## CHAPTER 6. SCALING CLUSTERS

### 6.1. SCALING KAFKA CLUSTERS

#### 6.1.1. Adding brokers to a cluster

The primary way of increasing throughput for a topic is to increase the number of partitions for that topic. That works because the partitions allow the load for that topic to be shared between the brokers in the cluster. When the brokers are all constrained by some resource (typically I/O), then using more partitions will not yield an increase in throughput. Instead, you must add brokers to the cluster.

When you add an extra broker to the cluster, AMQ Streams does not assign any partitions to it automatically. You have to decide which partitions to move from the existing brokers to the new broker.

Once the partitions have been redistributed between all brokers, each broker should have a lower resource utilization.

#### 6.1.2. Removing brokers from the cluster

Before you remove a broker from a cluster, you must ensure that it is not assigned to any partitions. You should decide which remaining brokers will be responsible for each of the partitions on the broker being decommissioned. Once the broker has no assigned partitions, you can stop it.

### 6.2. REASSIGNMENT OF PARTITIONS

The **kafka-reassign-partitions.sh** utility is used to reassign partitions to different brokers.

It has three different modes:

#### **--generate**

Takes a set of topics and brokers and generates a *reassignment JSON file* which will result in the partitions of those topics being assigned to those brokers. It is an easy way to generate a *reassignment JSON file*, but it operates on whole topics, so its use is not always appropriate.

#### **--execute**

Takes a *reassignment JSON file* and applies it to the partitions and brokers in the cluster. Brokers which are gaining partitions will become followers of the partition leader. For a given partition, once the new broker has caught up and joined the ISR the old broker will stop being a follower and will delete its replica.

#### **--verify**

Using the same *reassignment JSON file* as the **--execute** step, **--verify** checks whether all of the partitions in the file have been moved to their intended brokers. If the reassignment is complete it will also remove any [throttles](#) which are in effect. Unless removed, throttles will continue to affect the cluster even after the reassignment has finished.

It is only possible to have one reassignment running in the cluster at any given time, and it is not possible to cancel a running reassignment. If you need to cancel a reassignment you have to wait for it to complete and then perform another reassignment to revert the effects of the first one. The **kafka-reassign-partitions.sh** will print the reassignment JSON for this reversion as part of its output. Very large reassignments should be broken down into a number of smaller reassignments in case there is a need to stop in-progress reassignment.

### 6.2.1. Reassignment JSON file

The *reassignment JSON file* has a specific structure:

```
{
  "version": 1,
  "partitions": [
    <PartitionObjects>
  ]
}
```

Where *<PartitionObjects>* is a comma-separated list of objects like:

```
{
  "topic": <TopicName>,
  "partition": <Partition>,
  "replicas": [ <AssignedBrokerIds> ],
  "log_dirs": [<LogDirs>]
}
```

The **"log\_dirs"** property is optional and is used to move the partition to a specific log directory.

The following is an example reassignment JSON file that assigns topic **topic-a**, partition **4** to brokers **2, 4** and **7**, and topic **topic-b** partition **2** to brokers **1, 5** and **7**:

```
{
  "version": 1,
  "partitions": [
    {
      "topic": "topic-a",
      "partition": 4,
      "replicas": [2,4,7]
    },
    {
      "topic": "topic-b",
      "partition": 2,
      "replicas": [1,5,7]
    }
  ]
}
```

Partitions not included in the JSON are not changed.

### 6.2.2. Generating reassignment JSON files

The easiest way to assign all the partitions for a given set of topics to a given set of brokers is to generate a reassignment JSON file using the **kafka-reassign-partitions.sh --generate**, command.

```
bin/kafka-reassign-partitions.sh --zookeeper <ZooKeeper> --topics-to-move-json-file <TopicsFile> --
broker-list <BrokerList> --generate
```

The **<TopicsFile>** is a JSON file which lists the topics to move. It has the following structure:

```
{
```

```

"version": 1,
"topics": [
  <TopicObjects>
]
}

```

where *<TopicObjects>* is a comma-separated list of objects like:

```

{
  "topic": <TopicName>
}

```

For example to move all the partitions of **topic-a** and **topic-b** to brokers **4** and **7**

```

bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --topics-to-move-json-file topics-to-be-moved.json --broker-list 4,7 --generate

```

where **topics-to-be-moved.json** has contents:

```

{
  "version": 1,
  "topics": [
    { "topic": "topic-a"},
    { "topic": "topic-b"}
  ]
}

```

### 6.2.3. Creating reassignment JSON files manually

You can manually create the reassignment JSON file if you want to move specific partitions.

## 6.3. REASSIGNMENT THROTTLES

Reassigning partitions can be a slow process because it can require moving lots of data between brokers. To avoid this having a detrimental impact on clients it is possible to *throttle* the reassignment. Using a throttle can mean the reassignment takes longer. If the throttle is too low then the newly assigned brokers will not be able to keep up with records being published and the reassignment will never complete. If the throttle is too high then clients will be impacted. For example, for producers, this could manifest as higher than normal latency waiting for acknowledgement. For consumers, this could manifest as a drop in throughput caused by higher latency between polls.

## 6.4. SCALING UP A KAFKA CLUSTER

This procedure describes how to increase the number of brokers in a Kafka cluster.

### Prerequisites

- An existing Kafka cluster.
- A new machine with the AMQ broker [installed](#).
- A *reassignment JSON file* of how partitions should be reassigned to brokers in the enlarged cluster.

## Procedure

1. Create a configuration file for the new broker using the same settings as for the other brokers in your cluster, except for **broker.id** which should be a number that is not already used by any of the other brokers.
2. Start the new Kafka broker passing the configuration file you created in the previous step as the argument to the **kafka-server-start.sh** script:

```
su - kafka
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
```

3. Verify that the Kafka broker is running.

```
jcmd | grep Kafka
```

4. Repeat the above steps for each new broker.
5. Execute the partition reassignment using the **kafka-reassign-partitions.sh** command line tool.

```
kafka-reassign-partitions.sh --zookeeper <ZooKeeperHostAndPort> --reassignment-json-file
<ReassignmentJsonFile> --execute
```

If you are going to throttle replication you can also pass the **--throttle** option with an inter-broker throttled rate in bytes per second. For example:

```
kafka-reassign-partitions.sh --zookeeper zookeeper1:2181 --reassignment-json-file
reassignment.json --throttle 5000000 --execute
```

This command will print out two reassignment JSON objects. The first records the current assignment for the partitions being moved. You should save this to a file in case you need to revert the reassignment later on. The second JSON object is the target reassignment you have passed in your reassignment JSON file.

6. If you need to change the throttle during reassignment you can use the same command line with a different throttled rate. For example:

```
kafka-reassign-partitions.sh --zookeeper zookeeper1:2181 --reassignment-json-file
reassignment.json --throttle 10000000 --execute
```

7. Periodically verify whether the reassignment has completed using the **kafka-reassign-partitions.sh** command line tool. This is the same command as the previous step but with the **--verify** option instead of the **--execute** option.

```
kafka-reassign-partitions.sh --zookeeper <ZooKeeperHostAndPort> --reassignment-json-file
<ReassignmentJsonFile> --verify
```

For example:

```
kafka-reassign-partitions.sh --zookeeper zookeeper1:2181 --reassignment-json-file
reassignment.json --verify
```

8. The reassignment has finished when the **--verify** command reports each of the partitions being moved as completed successfully. This final **--verify** will also have the effect of removing any

reassignment throttles. You can now delete the revert file if you saved the JSON for reverting the assignment to their original brokers.

## 6.5. SCALING DOWN A KAFKA CLUSTER

### Additional resources

This procedure describes how to decrease the number of brokers in a Kafka cluster.

### Prerequisites

- An existing Kafka cluster.
- A *reassignment JSON file* of how partitions should be reassigned to brokers in the cluster once the broker(s) have been removed.

### Procedure

1. Execute the partition reassignment using the **kafka-reassign-partitions.sh** command line tool.

```
kafka-reassign-partitions.sh --zookeeper <ZooKeeperHostAndPort> --reassignment-json-file <ReassignmentJsonFile> --execute
```

If you are going to throttle replication you can also pass the **--throttle** option with an inter-broker throttled rate in bytes per second. For example:

```
kafka-reassign-partitions.sh --zookeeper zookeeper1:2181 --reassignment-json-file reassignment.json --throttle 5000000 --execute
```

This command will print out two reassignment JSON objects. The first records the current assignment for the partitions being moved. You should save this to a file in case you need to revert the reassignment later on. The second JSON object is the target reassignment you have passed in your reassignment JSON file.

2. If you need to change the throttle during reassignment you can use the same command line with a different throttled rate. For example:

```
kafka-reassign-partitions.sh --zookeeper zookeeper1:2181 --reassignment-json-file reassignment.json --throttle 10000000 --execute
```

3. Periodically verify whether the reassignment has completed using the **kafka-reassign-partitions.sh** command line tool. This is the same command as the previous step but with the **--verify** option instead of the **--execute** option.

```
kafka-reassign-partitions.sh --zookeeper <ZooKeeperHostAndPort> --reassignment-json-file <ReassignmentJsonFile> --verify
```

For example:

```
kafka-reassign-partitions.sh --zookeeper zookeeper1:2181 --reassignment-json-file reassignment.json --verify
```

4. The reassignment has finished when the **--verify** command reports each of the partitions being

moved as completed successfully. This final **--verify** will also have the effect of removing any reassignment throttles. You can now delete the revert file if you saved the JSON for reverting the assignment to their original brokers.

- Once all the partition reassignments have finished, the broker being removed should not have responsibility for any of the partitions in the cluster. You can verify this by checking each of the directories given in the broker's **log.dirs** configuration parameters. If any of the log directories on the broker contains a directory that does not match the extended regular expression **\.[a-z0-9]-delete\$** then the broker still has live partitions and it should not be stopped. You can check this by executing the command:

```
ls -l <LogDir> | grep -E '^d' | grep -vE '[a-zA-Z0-9.-]+\.[a-z0-9]+-delete$'
```

If the above command prints any output then the broker still has live partitions. In this case, either the reassignment has not finished, or the reassignment JSON file was incorrect.

- Once you have confirmed that the broker has no live partitions you can stop it.

```
su - kafka
/opt/kafka/bin/kafka-server-stop.sh
```

- Confirm that the Kafka broker is stopped.

```
jcmd | grep kafka
```

## 6.6. SCALING UP A ZOOKEEPER CLUSTER

This procedure describes how to add servers (nodes) to a ZooKeeper cluster. The [dynamic reconfiguration](#) feature of ZooKeeper maintains a stable ZooKeeper cluster during the scale up process.

### Prerequisites

- Dynamic reconfiguration is enabled in the ZooKeeper configuration file (**reconfigEnabled=true**).
- ZooKeeper authentication is enabled and you can access the new server using the authentication mechanism.

### Procedure

Perform the following steps for each ZooKeeper server, one at a time:

- Add a server to the ZooKeeper cluster as described in [Section 3.3, “Running multi-node ZooKeeper cluster”](#) and then start ZooKeeper.
- Note the IP address and configured access ports of the new server.
- Start a **zookeeper-shell** session for the server. Run the following command from a machine that has access to the cluster (this might be one of the ZooKeeper nodes or your local machine, if it has access).

```
su - kafka
/opt/kafka/bin/zookeeper-shell.sh <ip-address>:<zk-port>
```

4. In the shell session, with the ZooKeeper node running, enter the following line to add the new server to the quorum as a voting member:

```
reconfig -add server.<positive-id> = <address1>:<port1>:<port2>[:role];[<client-port-  
address>:]<client-port>
```

For example:

```
reconfig -add server.4=172.17.0.4:2888:3888:participant;172.17.0.4:2181
```

Where **<positive-id>** is the new server ID **4**.

For the two ports, **<port1>** 2888 is for communication between ZooKeeper servers, and **<port2>** 3888 is for leader election.

The new configuration propagates to the other servers in the ZooKeeper cluster; the new server is now a full member of the quorum.

5. Repeat steps 1-4 for the other servers that you want to add.

### Additional resources

- [Section 6.7, "Scaling down a ZooKeeper cluster"](#)

## 6.7. SCALING DOWN A ZOOKEEPER CLUSTER

This procedure describes how to remove servers (nodes) from a ZooKeeper cluster. The [dynamic reconfiguration](#) feature of ZooKeeper maintains a stable ZooKeeper cluster during the scale down process.

### Prerequisites

- Dynamic reconfiguration is enabled in the ZooKeeper configuration file (**reconfigEnabled=true**).
- ZooKeeper authentication is enabled and you can access the new server using the authentication mechanism.

### Procedure

Perform the following steps for each ZooKeeper server, one at a time:

1. Log in to the **zookeeper-shell** on one of the servers that will be **retained** after the scale down (for example, server 1).



#### NOTE

Access the server using the authentication mechanism configured for the ZooKeeper cluster.

2. Remove a server, for example server 5.

```
reconfig -remove 5
```



3. Deactivate the server that you removed.
4. Repeat steps 1-3 to reduce the cluster size.

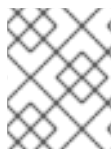
**Additional resources**

- [Section 6.6, "Scaling up a ZooKeeper cluster"](#)
- [Removing servers](#) in the ZooKeeper documentation

## CHAPTER 7. MONITORING YOUR CLUSTER USING JMX

ZooKeeper, the Kafka broker, Kafka Connect, and the Kafka clients all expose management information using [Java Management Extensions](#) (JMX). Most management information is in the form of metrics that are useful for monitoring the condition and performance of your Kafka cluster. Like other Java applications, Kafka provides this management information through managed beans or MBeans.

JMX works at the level of the JVM (Java Virtual Machine). To obtain management information, external tools can connect to the JVM that is running ZooKeeper, the Kafka broker, and so on. By default, only tools on the same machine and running as the same user as the JVM are able to connect.



### NOTE

Management information for ZooKeeper is not documented here. You can view ZooKeeper metrics in JConsole. For more information, see [Monitoring using JConsole](#).

### 7.1. JMX CONFIGURATION OPTIONS

You configure JMX using JVM system properties. The scripts provided with AMQ Streams (**bin/kafka-server-start.sh** and **bin/connect-distributed.sh**, and so on) use the **KAFKA\_JMX\_OPTS** environment variable to set these system properties. The system properties for configuring JMX are the same, even though Kafka producer, consumer, and streams applications typically start the JVM in different ways.

### 7.2. DISABLING THE JMX AGENT

You can prevent local JMX tools from connecting to the JVM (for example, for compliance reasons) by disabling the JMX agent for an AMQ Streams component. The following procedure explains how to disable the JMX agent for a Kafka broker.

#### Procedure

1. Use the **KAFKA\_JMX\_OPTS** environment variable to set **com.sun.management.jmxremote** to **false**.

```
export KAFKA_JMX_OPTS=-Dcom.sun.management.jmxremote=false
bin/kafka-server-start.sh
```

2. Start the JVM.

### 7.3. CONNECTING TO THE JVM FROM A DIFFERENT MACHINE

You can connect to the JVM from a different machine by configuring the port that the JMX agent listens on. This is insecure because it allows JMX tools to connect from anywhere, with no authentication.

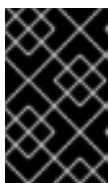
#### Procedure

1. Use the **KAFKA\_JMX\_OPTS** environment variable to set - **Dcom.sun.management.jmxremote.port=<port>**. For **<port>**, enter the name of the port on which you want the Kafka broker to listen for JMX connections.

```
export KAFKA_JMX_OPTS="-Dcom.sun.management.jmxremote=true
-Dcom.sun.management.jmxremote.port=<port>
```

```
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false"
bin/kafka-server-start.sh
```

2. Start the JVM.



### IMPORTANT

It is recommended that you configure authentication and SSL to ensure that the remote JMX connection is secure. For more information about the system properties needed to do this, see the [JMX documentation](#).

## 7.4. MONITORING USING JCONSOLE

The JConsole tool is distributed with the Java Development Kit (JDK). You can use JConsole to connect to a local or remote JVM and discover and display management information from Java applications. If using JConsole to connect to a local JVM, the names of the JVM processes corresponding to the different components of AMQ Streams are as follows:

AMQ Streams component	JVM process
ZooKeeper	<b>org.apache.zookeeper.server.quorum.QuorumPeerMain</b>
Kafka broker	<b>kafka.Kafka</b>
Kafka Connect standalone	<b>org.apache.kafka.connect.cli.ConnectStandalone</b>
Kafka Connect distributed	<b>org.apache.kafka.connect.cli.ConnectDistributed</b>
A Kafka producer, consumer, or Streams application	The name of the class containing the <b>main</b> method for the application.

When using JConsole to connect to a remote JVM, use the appropriate host name and JMX port.

Many other tools and monitoring products can be used to fetch the metrics using JMX and provide monitoring and alerting based on those metrics. Refer to the product documentation for those tools.

## 7.5. IMPORTANT KAFKA BROKER METRICS

Kafka provides many MBeans for monitoring the performance of the brokers in your Kafka cluster. These apply to an individual broker rather than the entire cluster.

The following tables present a selection of these broker-level MBeans organized into server, network, logging, and controller metrics.

### 7.5.1. Kafka server metrics

The following table shows a selection of metrics that report information about the Kafka server.

Metric	MBean	Description	Expected value
Messages in per second	<b>kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec</b>	The rate at which individual messages are consumed by the broker.	Approximately the same as the other brokers in the cluster.
Bytes in per second	<b>kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec</b>	The rate at which data sent from producers is consumed by the broker.	Approximately the same as the other brokers in the cluster.
Replication bytes in per second	<b>kafka.server:type=BrokerTopicMetrics,name=ReplicationBytesInPerSec</b>	The rate at which data sent from other brokers is consumed by the follower broker.	N/A
Bytes out per second	<b>kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec</b>	The rate at which data is fetched and read from the broker by consumers.	N/A
Replication bytes out per second	<b>kafka.server:type=BrokerTopicMetrics,name=ReplicationBytesOutPerSec</b>	The rate at which data is sent from the broker to other brokers. This metric is useful to monitor if the broker is a leader for a group of partitions.	N/A
Under-replicated partitions	<b>kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions</b>	The number of partitions that have not been fully replicated in the follower replicas.	Zero
Under minimum ISR partition count	<b>kafka.server:type=ReplicaManager,name=UnderMinIsrPartitionCount</b>	The number of partitions under the minimum In-Sync Replica (ISR) count. The ISR count indicates the set of replicas that are up-to-date with the leader.	Zero
Partition count	<b>kafka.server:type=ReplicaManager,name=PartitionCount</b>	The number of partitions in the broker.	Approximately even when compared with the other brokers.
Leader count	<b>kafka.server:type=ReplicaManager,name=LeaderCount</b>	The number of replicas for which this broker is the leader.	Approximately the same as the other brokers in the cluster.

Metric	MBean	Description	Expected value
ISR shrinks per second	<b>kafka.server:type=ReplicaManager,name=IsrShrinksPerSec</b>	The rate at which the number of ISRs in the broker decreases	Zero
ISR expands per second	<b>kafka.server:type=ReplicaManager,name=IsrExpandsPerSec</b>	The rate at which the number of ISRs in the broker increases.	Zero
Maximum lag	<b>kafka.server:type=ReplicaFetcherManager,name=MaxLag,clientId=Replica</b>	The maximum lag between the time that messages are received by the leader replica and by the follower replicas.	Proportional to the maximum batch size of a produce request.
Requests in producer purgatory	<b>kafka.server:type=DelayedOperationPurgatory,name=PurgatorySize,delayedOperation=Produce</b>	The number of send requests in the producer purgatory.	N/A
Requests in fetch purgatory	<b>kafka.server:type=DelayedOperationPurgatory,name=PurgatorySize,delayedOperation=Fetch</b>	The number of fetch requests in the fetch purgatory.	N/A
Request handler average idle percent	<b>kafka.server:type=KafkaRequestHandlerPool,name=RequestHandlerAvgIdlePercent</b>	Indicates the percentage of time that the request handler (IO) threads are not in use.	A lower value indicates that the workload of the broker is high.
Request (Requests exempt from throttling)	<b>kafka.server:type=Request</b>	The number of requests that are exempt from throttling.	N/A
ZooKeeper request latency in milliseconds	<b>kafka.server:type=ZooKeeperClientMetrics,name=ZooKeeperRequestLatencyMs</b>	The latency for ZooKeeper requests from the broker, in milliseconds.	N/A
ZooKeeper session state	<b>kafka.server:type=SessionExpireListener,name=SessionState</b>	The status of the broker's connection to ZooKeeper.	CONNECTED

## 7.5.2. Kafka network metrics

The following table shows a selection of metrics that report information about requests.

Metric	MBean	Description	Expected value
Requests per second	<b>kafka.network:type=RequestMetrics,name=RequestsPerSec,request={Produce FetchConsumer FetchFollower}</b>	The total number of requests made for the request type per second. The <b>Produce</b> , <b>FetchConsumer</b> , and <b>FetchFollower</b> request types each have their own MBeans.	N/A
Request bytes (request size in bytes)	<b>kafka.network:type=RequestMetrics,name=RequestBytes,request={[-.\w]+}</b>	The size of requests, in bytes, made for the request type identified by the <b>request</b> property of the MBean name. Separate MBeans for all available request types are listed under the <b>RequestBytes</b> node.	N/A
Temporary memory size in bytes	<b>kafka.network:type=RequestMetrics,name=TemporaryMemoryBytes,request={Produce Fetch}</b>	The amount of temporary memory used for converting message formats and decompressing messages.	N/A
Message conversions time	<b>kafka.network:type=RequestMetrics,name=MessageConversionsTimeMs,request={Produce Fetch}</b>	Time, in milliseconds, spent on converting message formats.	N/A
Total request time in milliseconds	<b>kafka.network:type=RequestMetrics,name=TotalTimeMs,request={Produce FetchConsumer FetchFollower}</b>	Total time, in milliseconds, spent processing requests.	N/A
Request queue time in milliseconds	<b>kafka.network:type=RequestMetrics,name=RequestQueueTimeMs,request={Produce FetchConsumer FetchFollower}</b>	The time, in milliseconds, that a request currently spends in the queue for the request type given in the <b>request</b> property.	N/A

Metric	MBean	Description	Expected value
Local time (leader local processing time) in milliseconds	<b>kafka.network:type=RequestMetrics,name=LocalTimeMs,request={Produce FetchConsumer FetchFollower}</b>	The time taken, in milliseconds, for the leader to process the request.	N/A
Remote time (leader remote processing time) in milliseconds	<b>kafka.network:type=RequestMetrics,name=RemoteTimeMs,request={Produce FetchConsumer FetchFollower}</b>	The length of time, in milliseconds, that the request waits for the follower. Separate MBeans for all available request types are listed under the <b>RemoteTimeMs</b> node.	N/A
Response queue time in milliseconds	<b>kafka.network:type=RequestMetrics,name=ResponseQueueTimeMs,request={Produce FetchConsumer FetchFollower}</b>	The length of time, in milliseconds, that the request waits in the response queue.	N/A
Response send time in milliseconds	<b>kafka.network:type=RequestMetrics,name=ResponseSendTimeMs,request={Produce FetchConsumer FetchFollower}</b>	The time taken, in milliseconds, to send the response.	N/A
Network processor average idle percent	<b>kafka.network:type=SocketServer,name=NetworkProcessorAvgIdlePercent</b>	The average percentage of time that the network processors are idle.	Between zero and one.

### 7.5.3. Kafka log metrics

The following table shows a selection of metrics that report information about logging.

Metric	MBean	Description	Expected Value
Log flush rate and time in milliseconds	<b>kafka.log:type=LogFlushStats,name=LogFlushRateAndTimeMs</b>	The rate at which log data is written to disk, in milliseconds.	N/A

Metric	MBean	Description	Expected Value
Offline log directory count	<b>kafka.log:type=LogManager,name=OfflineLogDirectoryCount</b>	The number of offline log directories (for example, after a hardware failure).	Zero

#### 7.5.4. Kafka controller metrics

The following table shows a selection of metrics that report information about the controller of the cluster.

Metric	MBean	Description	Expected Value
Active controller count	<b>kafka.controller:type=KafkaController,name=ActiveControllerCount</b>	The number of brokers designated as controllers.	One indicates that the broker is the controller for the cluster.
Leader election rate and time in milliseconds	<b>kafka.controller:type=ControllerStats,name=LeaderElectionRateAndTimeMs</b>	The rate at which new leader replicas are elected.	Zero

#### 7.5.5. Yammer metrics

Metrics that express a rate or unit of time are provided as Yammer metrics. The class name of an MBean that uses Yammer metrics is prefixed with **com.yammer.metrics**.

Yammer rate metrics have the following attributes for monitoring requests:

- Count
- EventType (Bytes)
- FifteenMinuteRate
- RateUnit (Seconds)
- MeanRate
- OneMinuteRate
- FiveMinuteRate

Yammer time metrics have the following attributes for monitoring requests:

- Max
- Min



- Mean
- StdDev
- 75/95/98/99/99.9<sup>th</sup> Percentile

## 7.6. PRODUCER MBEANS

The following MBeans will exist in Kafka producer applications, including Kafka Streams applications and Kafka Connect with source connectors.

### 7.6.1. MBeans matching `kafka.producer:type=producer-metrics,client-id=*`

These are metrics at the producer level.

Attribute	Description
batch-size-avg	The average number of bytes sent per partition per-request.
batch-size-max	The max number of bytes sent per partition per-request.
batch-split-rate	The average number of batch splits per second.
batch-split-total	The total number of batch splits.
buffer-available-bytes	The total amount of buffer memory that is not being used (either unallocated or in the free list).
buffer-total-bytes	The maximum amount of buffer memory the client can use (whether or not it is currently used).
bufferpool-wait-time	The fraction of time an appender waits for space allocation.
compression-rate-avg	The average compression rate of record batches.
connection-close-rate	Connections closed per second in the window.
connection-count	The current number of active connections.
connection-creation-rate	New connections established per second in the window.
failed-authentication-rate	Connections that failed authentication.
incoming-byte-rate	Bytes/second read off all sockets.

Attribute	Description
io-ratio	The fraction of time the I/O thread spent doing I/O.
io-time-ns-avg	The average length of time for I/O per select call in nanoseconds.
io-wait-ratio	The fraction of time the I/O thread spent waiting.
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.
metadata-age	The age in seconds of the current producer metadata being used.
network-io-rate	The average number of network operations (reads or writes) on all connections per second.
outgoing-byte-rate	The average number of outgoing bytes sent per second to all servers.
produce-throttle-time-avg	The average time in ms a request was throttled by a broker.
produce-throttle-time-max	The maximum time in ms a request was throttled by a broker.
record-error-rate	The average per-second number of record sends that resulted in errors.
record-error-total	The total number of record sends that resulted in errors.
record-queue-time-avg	The average time in ms record batches spent in the send buffer.
record-queue-time-max	The maximum time in ms record batches spent in the send buffer.
record-retry-rate	The average per-second number of retried record sends.
record-retry-total	The total number of retried record sends.
record-send-rate	The average number of records sent per second.
record-send-total	The total number of records sent.

Attribute	Description
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.
request-latency-avg	The average request latency in ms.
request-latency-max	The maximum request latency in ms.
request-rate	The average number of requests sent per second.
request-size-avg	The average size of all requests in the window.
request-size-max	The maximum size of any request sent in the window.
requests-in-flight	The current number of in-flight requests awaiting a response.
response-rate	Responses received sent per second.
select-rate	Number of times the I/O layer checked for new I/O to perform per second.
successful-authentication-rate	Connections that were successfully authenticated using SASL or SSL.
waiting-threads	The number of user threads blocked waiting for buffer memory to enqueue their records.

### 7.6.2. MBeans matching `kafka.producer:type=producer-metrics,client-id=*,node-id=*`

These are metrics at the producer level about connection to each broker.

Attribute	Description
incoming-byte-rate	The average number of responses received per second for a node.
outgoing-byte-rate	The average number of outgoing bytes sent per second for a node.
request-latency-avg	The average request latency in ms for a node.

Attribute	Description
request-latency-max	The maximum request latency in ms for a node.
request-rate	The average number of requests sent per second for a node.
request-size-avg	The average size of all requests in the window for a node.
request-size-max	The maximum size of any request sent in the window for a node.
response-rate	Responses received sent per second for a node.

### 7.6.3. MBeans matching `kafka.producer:type=producer-topic-metrics,client-id=*,topic=*`

These are metrics at the topic level about topics the producer is sending messages to.

Attribute	Description
byte-rate	The average number of bytes sent per second for a topic.
byte-total	The total number of bytes sent for a topic.
compression-rate	The average compression rate of record batches for a topic.
record-error-rate	The average per-second number of record sends that resulted in errors for a topic.
record-error-total	The total number of record sends that resulted in errors for a topic.
record-retry-rate	The average per-second number of retried record sends for a topic.
record-retry-total	The total number of retried record sends for a topic.
record-send-rate	The average number of records sent per second for a topic.
record-send-total	The total number of records sent for a topic.

## 7.7. CONSUMER MBEANS

The following MBeans will exist in Kafka consumer applications, including Kafka Streams applications and Kafka Connect with sink connectors.

### 7.7.1. MBeans matching `kafka.consumer:type=consumer-metrics,client-id=*`

These are metrics at the consumer level.

Attribute	Description
connection-close-rate	Connections closed per second in the window.
connection-count	The current number of active connections.
connection-creation-rate	New connections established per second in the window.
failed-authentication-rate	Connections that failed authentication.
incoming-byte-rate	Bytes/second read off all sockets.
io-ratio	The fraction of time the I/O thread spent doing I/O.
io-time-ns-avg	The average length of time for I/O per select call in nanoseconds.
io-wait-ratio	The fraction of time the I/O thread spent waiting.
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.
network-io-rate	The average number of network operations (reads or writes) on all connections per second.
outgoing-byte-rate	The average number of outgoing bytes sent per second to all servers.
request-rate	The average number of requests sent per second.
request-size-avg	The average size of all requests in the window.
request-size-max	The maximum size of any request sent in the window.
response-rate	Responses received sent per second.
select-rate	Number of times the I/O layer checked for new I/O to perform per second.

Attribute	Description
successful-authentication-rate	Connections that were successfully authenticated using SASL or SSL.

### 7.7.2. MBeans matching `kafka.consumer:type=consumer-metrics,client-id=*,node-id=*`

These are metrics at the consumer level about connection to each broker.

Attribute	Description
incoming-byte-rate	The average number of responses received per second for a node.
outgoing-byte-rate	The average number of outgoing bytes sent per second for a node.
request-latency-avg	The average request latency in ms for a node.
request-latency-max	The maximum request latency in ms for a node.
request-rate	The average number of requests sent per second for a node.
request-size-avg	The average size of all requests in the window for a node.
request-size-max	The maximum size of any request sent in the window for a node.
response-rate	Responses received sent per second for a node.

### 7.7.3. MBeans matching `kafka.consumer:type=consumer-coordinator-metrics,client-id=*`

These are metrics at the consumer level about the consumer group.

Attribute	Description
assigned-partitions	The number of partitions currently assigned to this consumer.
commit-latency-avg	The average time taken for a commit request.
commit-latency-max	The max time taken for a commit request.
commit-rate	The number of commit calls per second.

Attribute	Description
heartbeat-rate	The average number of heartbeats per second.
heartbeat-response-time-max	The max time taken to receive a response to a heartbeat request.
join-rate	The number of group joins per second.
join-time-avg	The average time taken for a group rejoin.
join-time-max	The max time taken for a group rejoin.
last-heartbeat-seconds-ago	The number of seconds since the last controller heartbeat.
sync-rate	The number of group syncs per second.
sync-time-avg	The average time taken for a group sync.
sync-time-max	The max time taken for a group sync.

#### 7.7.4. MBeans matching `kafka.consumer:type=consumer-fetch-manager-metrics,client-id=*`

These are metrics at the consumer level about the consumer's fetcher.

Attribute	Description
bytes-consumed-rate	The average number of bytes consumed per second.
bytes-consumed-total	The total number of bytes consumed.
fetch-latency-avg	The average time taken for a fetch request.
fetch-latency-max	The max time taken for any fetch request.
fetch-rate	The number of fetch requests per second.
fetch-size-avg	The average number of bytes fetched per request.
fetch-size-max	The maximum number of bytes fetched per request.
fetch-throttle-time-avg	The average throttle time in ms.
fetch-throttle-time-max	The maximum throttle time in ms.

Attribute	Description
fetch-total	The total number of fetch requests.
records-consumed-rate	The average number of records consumed per second.
records-consumed-total	The total number of records consumed.
records-lag-max	The maximum lag in terms of number of records for any partition in this window.
records-lead-min	The minimum lead in terms of number of records for any partition in this window.
records-per-request-avg	The average number of records in each request.

### 7.7.5. MBeans matching `kafka.consumer:type=consumer-fetch-manager-metrics,client-id=*,topic=*`

These are metrics at the topic level about the consumer's fetcher.

Attribute	Description
bytes-consumed-rate	The average number of bytes consumed per second for a topic.
bytes-consumed-total	The total number of bytes consumed for a topic.
fetch-size-avg	The average number of bytes fetched per request for a topic.
fetch-size-max	The maximum number of bytes fetched per request for a topic.
records-consumed-rate	The average number of records consumed per second for a topic.
records-consumed-total	The total number of records consumed for a topic.
records-per-request-avg	The average number of records in each request for a topic.

### 7.7.6. MBeans matching `kafka.consumer:type=consumer-fetch-manager-metrics,client-id=*,topic=*,partition=*`

These are metrics at the partition level about the consumer's fetcher.



Attribute	Description
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.

## 7.8. KAFKA CONNECT MBEANS



### NOTE

Kafka Connect will contain the [producer](#) MBeans for source connectors and [consumer](#) MBeans for sink connectors in addition to those documented here.

### 7.8.1. MBeans matching `kafka.connect:type=connect-metrics,client-id=*`

These are metrics at the connect level.

Attribute	Description
connection-close-rate	Connections closed per second in the window.
connection-count	The current number of active connections.
connection-creation-rate	New connections established per second in the window.
failed-authentication-rate	Connections that failed authentication.
incoming-byte-rate	Bytes/second read off all sockets.
io-ratio	The fraction of time the I/O thread spent doing I/O.
io-time-ns-avg	The average length of time for I/O per select call in nanoseconds.
io-wait-ratio	The fraction of time the I/O thread spent waiting.

Attribute	Description
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.
network-io-rate	The average number of network operations (reads or writes) on all connections per second.
outgoing-byte-rate	The average number of outgoing bytes sent per second to all servers.
request-rate	The average number of requests sent per second.
request-size-avg	The average size of all requests in the window.
request-size-max	The maximum size of any request sent in the window.
response-rate	Responses received sent per second.
select-rate	Number of times the I/O layer checked for new I/O to perform per second.
successful-authentication-rate	Connections that were successfully authenticated using SASL or SSL.

### 7.8.2. MBeans matching `kafka.connect:type=connect-metrics,client-id=*,node-id=*`

These are metrics at the connect level about connection to each broker.

Attribute	Description
incoming-byte-rate	The average number of responses received per second for a node.
outgoing-byte-rate	The average number of outgoing bytes sent per second for a node.
request-latency-avg	The average request latency in ms for a node.
request-latency-max	The maximum request latency in ms for a node.
request-rate	The average number of requests sent per second for a node.

Attribute	Description
request-size-avg	The average size of all requests in the window for a node.
request-size-max	The maximum size of any request sent in the window for a node.
response-rate	Responses received sent per second for a node.

### 7.8.3. MBeans matching `kafka.connect:type=connect-worker-metrics`

These are metrics at the connect level.

Attribute	Description
connector-count	The number of connectors run in this worker.
connector-startup-attempts-total	The total number of connector startups that this worker has attempted.
connector-startup-failure-percentage	The average percentage of this worker's connectors starts that failed.
connector-startup-failure-total	The total number of connector starts that failed.
connector-startup-success-percentage	The average percentage of this worker's connectors starts that succeeded.
connector-startup-success-total	The total number of connector starts that succeeded.
task-count	The number of tasks run in this worker.
task-startup-attempts-total	The total number of task startups that this worker has attempted.
task-startup-failure-percentage	The average percentage of this worker's tasks starts that failed.
task-startup-failure-total	The total number of task starts that failed.
task-startup-success-percentage	The average percentage of this worker's tasks starts that succeeded.
task-startup-success-total	The total number of task starts that succeeded.

### 7.8.4. MBeans matching `kafka.connect:type=connect-worker-rebalance-metrics`

Attribute	Description
completed-rebalances-total	The total number of rebalances completed by this worker.
epoch	The epoch or generation number of this worker.
leader-name	The name of the group leader.
rebalance-avg-time-ms	The average time in milliseconds spent by this worker to rebalance.
rebalance-max-time-ms	The maximum time in milliseconds spent by this worker to rebalance.
rebalancing	Whether this worker is currently rebalancing.
time-since-last-rebalance-ms	The time in milliseconds since this worker completed the most recent rebalance.

### 7.8.5. MBeans matching `kafka.connect:type=connector-metrics,connector=*`

Attribute	Description
connector-class	The name of the connector class.
connector-type	The type of the connector. One of 'source' or 'sink'.
connector-version	The version of the connector class, as reported by the connector.
status	The status of the connector. One of 'unassigned', 'running', 'paused', 'failed', or 'destroyed'.

### 7.8.6. MBeans matching `kafka.connect:type=connector-task-metrics,connector=*,task=*`

Attribute	Description
batch-size-avg	The average size of the batches processed by the connector.
batch-size-max	The maximum size of the batches processed by the connector.

Attribute	Description
offset-commit-avg-time-ms	The average time in milliseconds taken by this task to commit offsets.
offset-commit-failure-percentage	The average percentage of this task's offset commit attempts that failed.
offset-commit-max-time-ms	The maximum time in milliseconds taken by this task to commit offsets.
offset-commit-success-percentage	The average percentage of this task's offset commit attempts that succeeded.
pause-ratio	The fraction of time this task has spent in the pause state.
running-ratio	The fraction of time this task has spent in the running state.
status	The status of the connector task. One of 'unassigned', 'running', 'paused', 'failed', or 'destroyed'.

### 7.8.7. MBeans matching `kafka.connect:type=sink-task-metrics,connector=*,task=*`

Attribute	Description
offset-commit-completion-rate	The average per-second number of offset commit completions that were completed successfully.
offset-commit-completion-total	The total number of offset commit completions that were completed successfully.
offset-commit-seq-no	The current sequence number for offset commits.
offset-commit-skip-rate	The average per-second number of offset commit completions that were received too late and skipped/ignored.
offset-commit-skip-total	The total number of offset commit completions that were received too late and skipped/ignored.
partition-count	The number of topic partitions assigned to this task belonging to the named sink connector in this worker.

Attribute	Description
put-batch-avg-time-ms	The average time taken by this task to put a batch of sinks records.
put-batch-max-time-ms	The maximum time taken by this task to put a batch of sinks records.
sink-record-active-count	The number of records that have been read from Kafka but not yet completely committed/flushed/acknowledged by the sink task.
sink-record-active-count-avg	The average number of records that have been read from Kafka but not yet completely committed/flushed/acknowledged by the sink task.
sink-record-active-count-max	The maximum number of records that have been read from Kafka but not yet completely committed/flushed/acknowledged by the sink task.
sink-record-lag-max	The maximum lag in terms of number of records that the sink task is behind the consumer's position for any topic partitions.
sink-record-read-rate	The average per-second number of records read from Kafka for this task belonging to the named sink connector in this worker. This is before transformations are applied.
sink-record-read-total	The total number of records read from Kafka by this task belonging to the named sink connector in this worker, since the task was last restarted.
sink-record-send-rate	The average per-second number of records output from the transformations and sent/put to this task belonging to the named sink connector in this worker. This is after transformations are applied and excludes any records filtered out by the transformations.
sink-record-send-total	The total number of records output from the transformations and sent/put to this task belonging to the named sink connector in this worker, since the task was last restarted.

### 7.8.8. MBeans matching `kafka.connect:type=source-task-metrics,connector=*,task=*`

Attribute	Description
-----------	-------------

Attribute	Description
poll-batch-avg-time-ms	The average time in milliseconds taken by this task to poll for a batch of source records.
poll-batch-max-time-ms	The maximum time in milliseconds taken by this task to poll for a batch of source records.
source-record-active-count	The number of records that have been produced by this task but not yet completely written to Kafka.
source-record-active-count-avg	The average number of records that have been produced by this task but not yet completely written to Kafka.
source-record-active-count-max	The maximum number of records that have been produced by this task but not yet completely written to Kafka.
source-record-poll-rate	The average per-second number of records produced/poll (before transformation) by this task belonging to the named source connector in this worker.
source-record-poll-total	The total number of records produced/poll (before transformation) by this task belonging to the named source connector in this worker.
source-record-write-rate	The average per-second number of records output from the transformations and written to Kafka for this task belonging to the named source connector in this worker. This is after transformations are applied and excludes any records filtered out by the transformations.
source-record-write-total	The number of records output from the transformations and written to Kafka for this task belonging to the named source connector in this worker, since the task was last restarted.

### 7.8.9. MBeans matching `kafka.connect:type=task-error-metrics,connector=*,task=*`

Attribute	Description
deadletterqueue-produce-failures	The number of failed writes to the dead letter queue.
deadletterqueue-produce-requests	The number of attempted writes to the dead letter queue.

Attribute	Description
last-error-timestamp	The epoch timestamp when this task last encountered an error.
total-errors-logged	The number of errors that were logged.
total-record-errors	The number of record processing errors in this task.
total-record-failures	The number of record processing failures in this task.
total-records-skipped	The number of records skipped due to errors.
total-retries	The number of operations retried.

## 7.9. KAFKA STREAMS MBEANS



### NOTE

A Streams application will contain the [producer](#) and [consumer](#) MBeans in addition to those documented here.

### 7.9.1. MBeans matching `kafka.streams:type=stream-metrics,client-id=*`

These metrics are collected when the `metrics.recording.level` configuration parameter is **info** or **debug**.

Attribute	Description
commit-latency-avg	The average execution time in ms for committing, across all running tasks of this thread.
commit-latency-max	The maximum execution time in ms for committing across all running tasks of this thread.
commit-rate	The average number of commits per second.
commit-total	The total number of commit calls across all tasks.
poll-latency-avg	The average execution time in ms for polling, across all running tasks of this thread.
poll-latency-max	The maximum execution time in ms for polling across all running tasks of this thread.
poll-rate	The average number of polls per second.



Attribute	Description
poll-total	The total number of poll calls across all tasks.
process-latency-avg	The average execution time in ms for processing, across all running tasks of this thread.
process-latency-max	The maximum execution time in ms for processing across all running tasks of this thread.
process-rate	The average number of process calls per second.
process-total	The total number of process calls across all tasks.
punctuate-latency-avg	The average execution time in ms for punctuating, across all running tasks of this thread.
punctuate-latency-max	The maximum execution time in ms for punctuating across all running tasks of this thread.
punctuate-rate	The average number of punctuates per second.
punctuate-total	The total number of punctuate calls across all tasks.
skipped-records-rate	The average number of skipped records per second.
skipped-records-total	The total number of skipped records.
task-closed-rate	The average number of tasks closed per second.
task-closed-total	The total number of tasks closed.
task-created-rate	The average number of newly created tasks per second.
task-created-total	The total number of tasks created.

### 7.9.2. MBeans matching `kafka.streams:type=stream-task-metrics,client-id=*,task-id=*`

Task metrics.

These metrics are collected when the `metrics.recording.level` configuration parameter is `debug`.

Attribute	Description
commit-latency-avg	The average commit time in ns for this task.

Attribute	Description
commit-latency-max	The maximum commit time in ns for this task.
commit-rate	The average number of commit calls per second.
commit-total	The total number of commit calls.

### 7.9.3. MBeans matching `kafka.streams:type=stream-processor-node-metrics,client-id=*,task-id=*,processor-node-id=*`

Processor node metrics.

These metrics are collected when the `metrics.recording.level` configuration parameter is `debug`.

Attribute	Description
create-latency-avg	The average create execution time in ns.
create-latency-max	The maximum create execution time in ns.
create-rate	The average number of create operations per second.
create-total	The total number of create operations called.
destroy-latency-avg	The average destroy execution time in ns.
destroy-latency-max	The maximum destroy execution time in ns.
destroy-rate	The average number of destroy operations per second.
destroy-total	The total number of destroy operations called.
forward-rate	The average rate of records being forwarded downstream, from source nodes only, per second.
forward-total	The total number of of records being forwarded downstream, from source nodes only.
process-latency-avg	The average process execution time in ns.
process-latency-max	The maximum process execution time in ns.
process-rate	The average number of process operations per second.

Attribute	Description
process-total	The total number of process operations called.
punctuate-latency-avg	The average punctuate execution time in ns.
punctuate-latency-max	The maximum punctuate execution time in ns.
punctuate-rate	The average number of punctuate operations per second.
punctuate-total	The total number of punctuate operations called.

#### 7.9.4. MBeans matching `kafka.streams:type=stream-[store-scope]-metrics,client-id=*,task-id=*,[store-scope]-id=*`

State store metrics.

These metrics are collected when the `metrics.recording.level` configuration parameter is `debug`.

Attribute	Description
all-latency-avg	The average all operation execution time in ns.
all-latency-max	The maximum all operation execution time in ns.
all-rate	The average all operation rate for this store.
all-total	The total number of all operation calls for this store.
delete-latency-avg	The average delete execution time in ns.
delete-latency-max	The maximum delete execution time in ns.
delete-rate	The average delete rate for this store.
delete-total	The total number of delete calls for this store.
flush-latency-avg	The average flush execution time in ns.
flush-latency-max	The maximum flush execution time in ns.
flush-rate	The average flush rate for this store.
flush-total	The total number of flush calls for this store.

Attribute	Description
get-latency-avg	The average get execution time in ns.
get-latency-max	The maximum get execution time in ns.
get-rate	The average get rate for this store.
get-total	The total number of get calls for this store.
put-all-latency-avg	The average put-all execution time in ns.
put-all-latency-max	The maximum put-all execution time in ns.
put-all-rate	The average put-all rate for this store.
put-all-total	The total number of put-all calls for this store.
put-if-absent-latency-avg	The average put-if-absent execution time in ns.
put-if-absent-latency-max	The maximum put-if-absent execution time in ns.
put-if-absent-rate	The average put-if-absent rate for this store.
put-if-absent-total	The total number of put-if-absent calls for this store.
put-latency-avg	The average put execution time in ns.
put-latency-max	The maximum put execution time in ns.
put-rate	The average put rate for this store.
put-total	The total number of put calls for this store.
range-latency-avg	The average range execution time in ns.
range-latency-max	The maximum range execution time in ns.
range-rate	The average range rate for this store.
range-total	The total number of range calls for this store.
restore-latency-avg	The average restore execution time in ns.
restore-latency-max	The maximum restore execution time in ns.
restore-rate	The average restore rate for this store.

Attribute	Description
restore-total	The total number of restore calls for this store.

### 7.9.5. MBeans matching `kafka.streams:type=stream-record-cache-metrics,client-id=*,task-id=*,record-cache-id=*`

Record cache metrics.

These metrics are collected when the `metrics.recording.level` configuration parameter is `debug`.

Attribute	Description
hitRatio-avg	The average cache hit ratio defined as the ratio of cache read hits over the total cache read requests.
hitRatio-max	The maximum cache hit ratio.
hitRatio-min	The minimum cache hit ratio.

## CHAPTER 8. KAFKA CONNECT

Kafka Connect is a tool for streaming data between Apache Kafka and external systems. It provides a framework for moving large amounts of data while maintaining scalability and reliability. Kafka Connect is typically used to integrate Kafka with database, storage, and messaging systems that are external to your Kafka cluster.

Kafka Connect uses connector plug-ins that implement connectivity for different types of external systems. There are two types of connector plug-ins: sink and source. Sink connectors stream data from Kafka to external systems. Source connectors stream data from external systems into Kafka.

Kafka Connect can run in standalone or distributed modes.

### Standalone mode

In standalone mode, Kafka Connect runs on a single node with user-defined configuration read from a properties file.

### Distributed mode

In distributed mode, Kafka Connect runs across one or more worker nodes and the workloads are distributed among them. You manage connectors and their configuration using an HTTP REST interface.

## 8.1. KAFKA CONNECT IN STANDALONE MODE

In standalone mode, Kafka Connect runs as a single process, on a single node. You manage the configuration of standalone mode using properties files.

### 8.1.1. Configuring Kafka Connect in standalone mode

To configure Kafka Connect in standalone mode, edit the **config/connect-standalone.properties** configuration file. The following options are the most important.

#### **bootstrap.servers**

A list of Kafka broker addresses used as bootstrap connections to Kafka. For example, **kafka0.my-domain.com:9092,kafka1.my-domain.com:9092,kafka2.my-domain.com:9092**.

#### **key.converter**

The class used to convert message keys to and from Kafka format. For example, **org.apache.kafka.connect.json.JsonConverter**.

#### **value.converter**

The class used to convert message payloads to and from Kafka format. For example, **org.apache.kafka.connect.json.JsonConverter**.

#### **offset.storage.file.filename**

Specifies the file in which the offset data is stored.

An example configuration file is provided in the installation directory at **config/connect-standalone.properties**. For a complete list of all supported Kafka Connect configuration options, see [kafka-connect-configuration-parameters-str].

Connector plug-ins open client connections to the Kafka brokers using the bootstrap address. To configure these connections, use the standard Kafka producer and consumer configuration options prefixed by **producer.** or **consumer.**

For more information on configuring Kafka producers and consumers, see:

- [Appendix D, Producer configuration parameters](#)
- [Appendix C, Consumer configuration parameters](#)

### 8.1.2. Configuring connectors in Kafka Connect in standalone mode

You can configure connector plug-ins for Kafka Connect in standalone mode using properties files. Most configuration options are specific to each connector. The following options apply to all connectors:

#### **name**

The name of the connector, which must be unique within the current Kafka Connect instance.

#### **connector.class**

The class of the connector plug-in. For example, **org.apache.kafka.connect.file.FileStreamSinkConnector**.

#### **tasks.max**

The maximum number of tasks that the specified connector can use. Tasks enable the connector to perform work in parallel. The connector might create fewer tasks than specified.

#### **key.converter**

The class used to convert message keys to and from Kafka format. This overrides the default value set by the Kafka Connect configuration. For example, **org.apache.kafka.connect.json.JsonConverter**.

#### **value.converter**

The class used to convert message payloads to and from Kafka format. This overrides the default value set by the Kafka Connect configuration. For example, **org.apache.kafka.connect.json.JsonConverter**.

Additionally, you must set at least one of the following options for sink connectors:

#### **topics**

A comma-separated list of topics used as input.

#### **topics.regex**

A Java regular expression of topics used as input.

For all other options, see the documentation for the available connectors.

AMQ Streams includes example connector configuration files – see **config/connect-file-sink.properties** and **config/connect-file-source.properties** in the AMQ Streams installation directory.

### 8.1.3. Running Kafka Connect in standalone mode

This procedure describes how to configure and run Kafka Connect in standalone mode.

#### **Prerequisites**

- An installed and running AMQ Streams} cluster.

#### **Procedure**

1. Edit the **/opt/kafka/config/connect-standalone.properties** Kafka Connect configuration file and set **bootstrap.server** to point to your Kafka brokers. For example:

```
bootstrap.servers=kafka0.my-domain.com:9092,kafka1.my-domain.com:9092,kafka2.my-domain.com:9092
```

2. Start Kafka Connect with the configuration file and specify one or more connector configurations.

```
su - kafka
/opt/kafka/bin/connect-standalone.sh /opt/kafka/config/connect-standalone.properties
connector1.properties
[connector2.properties ...]
```

3. Verify that Kafka Connect is running.

```
jcmd | grep ConnectStandalone
```

### Additional resources

- For more information on installing AMQ Streams, see [Section 2.3, “Installing AMQ Streams”](#).
- For more information on configuring AMQ Streams, see [Section 2.8, “Configuring AMQ Streams”](#).
- For a complete list of supported Kafka Connect configuration options, see [Appendix F, \*Kafka Connect configuration parameters\*](#).

## 8.2. KAFKA CONNECT IN DISTRIBUTED MODE

In distributed mode, Kafka Connect runs across one or more worker nodes and the workloads are distributed among them. You manage connector plug-ins and their configuration using the HTTP REST interface.

### 8.2.1. Configuring Kafka Connect in distributed mode

To configure Kafka Connect in distributed mode, edit the **config/connect-distributed.properties** configuration file. The following options are the most important.

#### **bootstrap.servers**

A list of Kafka broker addresses used as bootstrap connections to Kafka. For example, **kafka0.my-domain.com:9092,kafka1.my-domain.com:9092,kafka2.my-domain.com:9092**.

#### **key.converter**

The class used to convert message keys to and from Kafka format. For example, **org.apache.kafka.connect.json.JsonConverter**.

#### **value.converter**

The class used to convert message payloads to and from Kafka format. For example, **org.apache.kafka.connect.json.JsonConverter**.

#### **group.id**

The name of the distributed Kafka Connect cluster. This must be unique and must not conflict with another consumer group ID. The default value is **connect-cluster**.

#### **config.storage.topic**

The Kafka topic used to store connector configurations. The default value is **connect-configs**.



**offset.storage.topic**

The Kafka topic used to store offsets. The default value is **connect-offset**.

**status.storage.topic**

The Kafka topic used for worker node statuses. The default value is **connect-status**.

AMQ Streams includes an example configuration file for Kafka Connect in distributed mode – see **config/connect-distributed.properties** in the AMQ Streams installation directory.

For a complete list of all supported Kafka Connect configuration options, see [Appendix F, Kafka Connect configuration parameters](#).

Connector plug-ins open client connections to the Kafka brokers using the bootstrap address. To configure these connections, use the standard Kafka producer and consumer configuration options prefixed by **producer.** or **consumer.**

For more information on configuring Kafka producers and consumers, see:

- [Appendix D, Producer configuration parameters](#)
- [Appendix C, Consumer configuration parameters](#)

**8.2.2. Configuring connectors in distributed Kafka Connect****HTTP REST Interface**

Connectors for distributed Kafka Connect are configured using HTTP REST interface. The REST interface listens on port 8083 by default. It supports following endpoints:

**GET /connectors**

Return a list of existing connectors.

**POST /connectors**

Create a connector. The request body has to be a JSON object with the connector configuration.

**GET /connectors/<name>**

Get information about a specific connector.

**GET /connectors/<name>/config**

Get configuration of a specific connector.

**PUT /connectors/<name>/config**

Update the configuration of a specific connector.

**GET /connectors/<name>/status**

Get the status of a specific connector.

**PUT /connectors/<name>/pause**

Pause the connector and all its tasks. The connector will stop processing any messages.

**PUT /connectors/<name>/resume**

Resume a paused connector.

**POST /connectors/<name>/restart**

Restart a connector in case it has failed.

**DELETE /connectors/<name>**

Delete a connector.

## GET /connector-plugins

Get a list of all supported connector plugins.

## Connector configuration

Most configuration options are connector specific and included in the documentation for the connectors. The following fields are common for all connectors.

### name

Name of the connector. Must be unique within a given Kafka Connect instance.

### connector.class

Class of the connector plugin. For example

**org.apache.kafka.connect.file.FileStreamSinkConnector.**

### tasks.max

The maximum number of tasks used by this connector. Tasks are used by the connector to parallelise its work. Connectors may create fewer tasks than specified.

### key.converter

Class used to convert message keys to and from Kafka format. This overrides the default value set by the Kafka Connect configuration. For example, **org.apache.kafka.connect.json.JsonConverter.**

### value.converter

Class used to convert message payloads to and from Kafka format. This overrides the default value set by the Kafka Connect configuration. For example,

**org.apache.kafka.connect.json.JsonConverter.**

Additionally, one of the following options must be set for sink connectors:

### topics

A comma-separated list of topics used as input.

### topics.regex

A Java regular expression of topics used as input.

For all other options, see the documentation for the specific connector.

AMQ Streams includes example connector configuration files. They can be found in **config/connect-file-sink.properties** and **config/connect-file-source.properties** in the AMQ Streams installation directory.

## 8.2.3. Running distributed Kafka Connect

This procedure describes how to configure and run Kafka Connect in distributed mode.

### Prerequisites

- An installed and running AMQ Streams cluster.

### Running the cluster

1. Edit the **/opt/kafka/config/connect-distributed.properties** Kafka Connect configuration file on all Kafka Connect worker nodes.
  - Set the **bootstrap.server** option to point to your Kafka brokers.

- Set the **group.id** option.
- Set the **config.storage.topic** option.
- Set the **offset.storage.topic** option.
- Set the **status.storage.topic** option.  
For example:

```
bootstrap.servers=kafka0.my-domain.com:9092,kafka1.my-domain.com:9092,kafka2.my-
domain.com:9092
group.id=my-group-id
config.storage.topic=my-group-id-configs
offset.storage.topic=my-group-id-offsets
status.storage.topic=my-group-id-status
```

2. Start the Kafka Connect workers with the **/opt/kafka/config/connect-distributed.properties** configuration file on all Kafka Connect nodes.

```
su - kafka
/opt/kafka/bin/connect-distributed.sh /opt/kafka/config/connect-distributed.properties
```

3. Verify that Kafka Connect is running.

```
jcmd | grep ConnectDistributed
```

### Additional resources

- For more information about installing AMQ Streams, see [Section 2.3, “Installing AMQ Streams”](#).
- For more information about configuring AMQ Streams, see [Section 2.8, “Configuring AMQ Streams”](#).
- For a complete list of supported Kafka Connect configuration options, see [Appendix F, \*Kafka Connect configuration parameters\*](#).

## 8.2.4. Creating connectors

This procedure describes how to use the Kafka Connect REST API to create a connector plug-in for use with Kafka Connect in distributed mode.

### Prerequisites

- A Kafka Connect installation running in distributed mode.

### Procedure

1. Prepare a JSON payload with the connector configuration. For example:

```
{
  "name": "my-connector",
  "config": {
    "connector.class": "org.apache.kafka.connect.file.FileStreamSinkConnector",
    "tasks.max": "1",
```

```

    "topics": "my-topic-1,my-topic-2",
    "file": "/tmp/output-file.txt"
  }
}

```

2. Send a POST request to **<KafkaConnectAddress>:8083/connectors** to create the connector. The following example uses **curl**:

```

curl -X POST -H "Content-Type: application/json" --data @sink-connector.json
http://connect0.my-domain.com:8083/connectors

```

3. Verify that the connector was deployed by sending a GET request to **<KafkaConnectAddress>:8083/connectors**. The following example uses **curl**:

```

curl http://connect0.my-domain.com:8083/connectors

```

### 8.2.5. Deleting connectors

This procedure describes how to use the Kafka Connect REST API to delete a connector plug-in from Kafka Connect in distributed mode.

#### Prerequisites

- A Kafka Connect installation running in distributed mode.

#### Deleting connectors

1. Verify that the connector exists by sending a **GET** request to **<KafkaConnectAddress>:8083/connectors/<ConnectorName>**. The following example uses **curl**:

```

curl http://connect0.my-domain.com:8083/connectors

```

2. To delete the connector, send a **DELETE** request to **<KafkaConnectAddress>:8083/connectors**. The following example uses **curl**:

```

curl -X DELETE http://connect0.my-domain.com:8083/connectors/my-connector

```

3. Verify that the connector was deleted by sending a GET request to **<KafkaConnectAddress>:8083/connectors**. The following example uses **curl**:

```

curl http://connect0.my-domain.com:8083/connectors

```

## 8.3. CONNECTOR PLUG-INS

The following connector plug-ins are included with AMQ Streams.

**FileStreamSink** Reads data from Kafka topics and writes the data to a file.

**FileStreamSource** Reads data from a file and sends the data to Kafka topics.

You can add more connector plug-ins if needed. Kafka Connect searches for and runs additional connector plug-ins at startup. To define the path that kafka Connect searches for plug-ins, set the **plugin.path configuration** option:

```
plugin.path=/opt/kafka/connector-plugins,/opt/connectors
```

The **plugin.path** configuration option can contain a comma-separated list of paths.

When running Kafka Connect in distributed mode, plug-ins must be made available on all worker nodes.

## 8.4. ADDING CONNECTOR PLUGINS

This procedure shows you how to add additional connector plug-ins.

### Prerequisites

- An installed and running AMQ Streams cluster.

### Procedure

1. Create the **/opt/kafka/connector-plugins** directory.

```
su - kafka
mkdir /opt/kafka/connector-plugins
```

2. Edit the **/opt/kafka/config/connect-standalone.properties** or **/opt/kafka/config/connect-distributed.properties** Kafka Connect configuration file, and set the **plugin.path** option to **/opt/kafka/connector-plugins**. For example:

```
plugin.path=/opt/kafka/connector-plugins
```

3. Copy your connector plug-ins to **/opt/kafka/connector-plugins**.
4. Start or restart the Kafka Connect workers.

### Additional resources

- For more information on installing AMQ Streams, see [Section 2.3, “Installing AMQ Streams”](#).
- For more information on configuring AMQ Streams, see [Section 2.8, “Configuring AMQ Streams”](#).
- For more information on running Kafka Connect in standalone mode, see [Section 8.1.3, “Running Kafka Connect in standalone mode”](#).
- For more information on running Kafka Connect in distributed mode, see [Section 8.2.3, “Running distributed Kafka Connect”](#).
- For a complete list of supported Kafka Connect configuration options, see [Appendix F, \*Kafka Connect configuration parameters\*](#).

## CHAPTER 9. USING AMQ STREAMS WITH MIRRORMAKER 2.0



### IMPORTANT

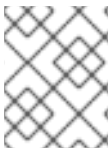
MirrorMaker 2.0 is a Technology Preview only. Technology Preview features are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend implementing any Technology Preview features in production environments. This Technology Preview feature provides early access to upcoming product innovations, enabling you to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

This section describes using AMQ Streams with MirrorMaker 2.0 as a Technology Preview. The previous version of MirrorMaker continues to be supported.

MirrorMaker is used to replicate data between two or more active Kafka clusters, within or across data centers.

Data replication across clusters supports scenarios that require:

- Recovery of data in the event of a system failure
- Aggregation of data for analysis
- Restriction of data access to a specific cluster
- Provision of data at a specific location to improve latency



### NOTE

MirrorMaker 2.0 has features not supported by the previous version of MirrorMaker. However, you can [configure MirrorMaker 2.0 to be used in legacy mode](#) .

### Additional resources

- [Apache Kafka documentation](#)

## 9.1. MIRRORMAKER 2.0 DATA REPLICATION

MirrorMaker 2.0 consumes messages from a source Kafka cluster and writes them to a target Kafka cluster.

MirrorMaker 2.0 uses:

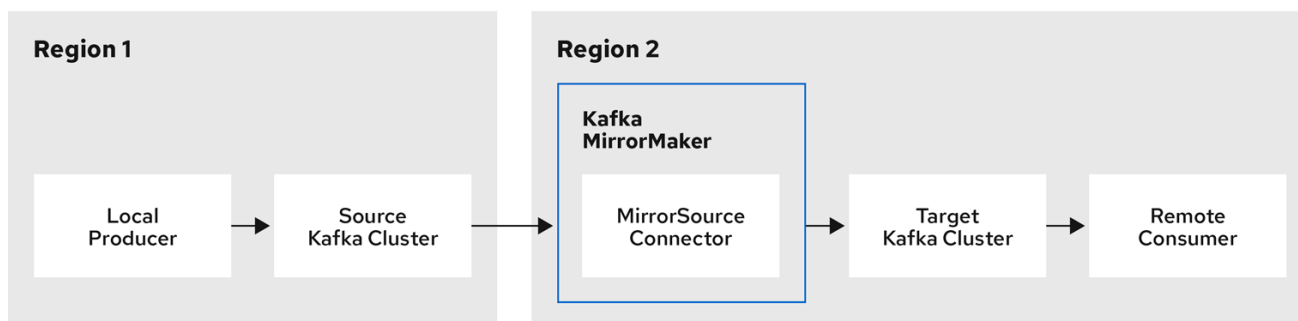
- Source cluster configuration to consume data from the source cluster
- Target cluster configuration to output data to the target cluster

MirrorMaker 2.0 is based on the Kafka Connect framework, *connectors* managing the transfer of data between clusters. A MirrorMaker 2.0 **MirrorSourceConnector** replicates topics from a source cluster to a target cluster.

The process of *mirroring* data from one cluster to another cluster is asynchronous. The recommended pattern is for messages to be produced locally alongside the source Kafka cluster, then consumed remotely close to the target Kafka cluster.

MirrorMaker 2.0 can be used with more than one source cluster.

Figure 9.1. Replication across two clusters



AMQ\_73\_0220

## 9.2. CLUSTER CONFIGURATION

You can use MirrorMaker 2.0 in *active/passive* or *active/active* cluster configurations.

- In an *active/passive* configuration, the data from an active cluster is replicated in a passive cluster, which remains on standby, for example, for data recovery in the event of system failure.
- In an *active/active* configuration, both clusters are active and provide the same data simultaneously, which is useful if you want to make the same data available locally in different geographical locations.

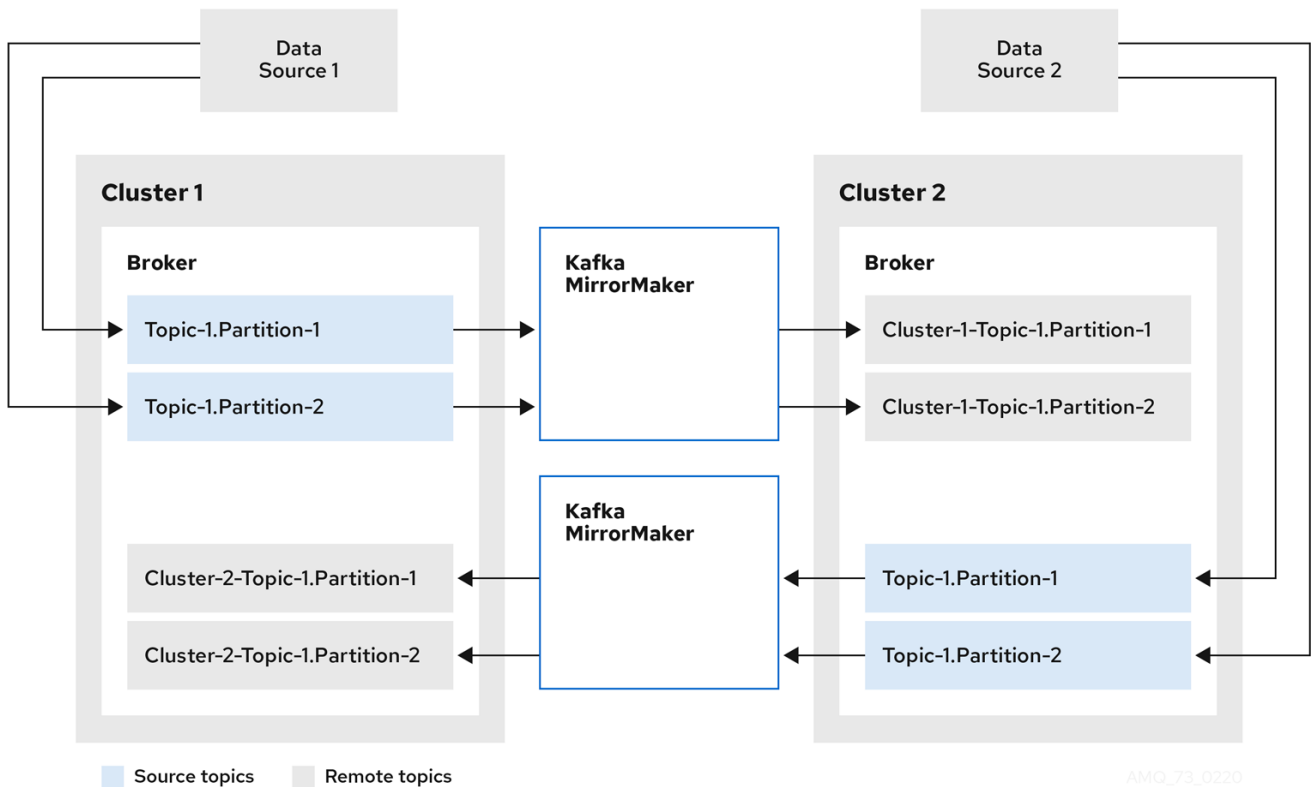
The expectation is that producers and consumers connect to active clusters only.

### 9.2.1. Bidirectional replication

The MirrorMaker 2.0 architecture supports bidirectional replication in an *active/active* cluster configuration. A MirrorMaker 2.0 cluster is required at each target destination.

Each cluster replicates the data of the other cluster using the concept of *source* and *remote* topics. As the same topics are stored in each cluster, remote topics are automatically renamed by MirrorMaker 2.0 to represent the source cluster.

Figure 9.2. Topic renaming



By flagging the originating cluster, topics are not replicated back to that cluster.

The concept of replication through *remote* topics is useful when configuring an architecture that requires data aggregation. Consumers can subscribe to source and remote topics within the same cluster, without the need for a separate aggregation cluster.

### 9.2.2. Topic configuration synchronization

Topic configuration is automatically synchronized between source and target clusters. By synchronizing configuration properties, the need for rebalancing is reduced.

### 9.2.3. Data integrity

MirrorMaker 2.0 monitors source topics and propagates any configuration changes to remote topics, checking for and creating missing partitions. Only MirrorMaker 2.0 can write to remote topics.

### 9.2.4. Offset tracking

MirrorMaker 2.0 tracks offsets for consumer groups using *internal topics*.

- The *offset sync* topic maps the source and target offsets for replicated topic partitions from record metadata
- The *checkpoint* topic maps the last committed offset in the source and target cluster for replicated topic partitions in each consumer group

Offsets for the *checkpoint* topic are tracked at predetermined intervals through configuration. Both topics enable replication to be fully restored from the correct offset position on failover.

MirrorMaker 2.0 uses its **MirrorCheckpointConnector** to emit *checkpoints* for offset tracking.



### 9.2.5. Connectivity checks

A *heartbeat* internal topic checks connectivity between clusters.

The *heartbeat* topic is replicated from the source cluster.

Target clusters use the topic to check:

- The connector managing connectivity between clusters is running
- The source cluster is available

MirrorMaker 2.0 uses its **MirrorHeartbeatConnector** to emit *heartbeats* that perform these checks.

## 9.3. ACL RULES SYNCHRONIZATION

If **SimpleAclAuthorizer** is being used, ACL rules that manage access to brokers also apply to remote topics. Users that can read a source topic can read its remote equivalent.



### NOTE

OAuth 2.0 authorization does not support access to remote topics in this way.

## 9.4. SYNCHRONIZING DATA BETWEEN KAFKA CLUSTERS USING MIRRORMAKER 2.0

Use MirrorMaker 2.0 to synchronize data between Kafka clusters through configuration.

The previous version of MirrorMaker continues to be supported, by [running MirrorMaker 2.0 in legacy mode](#).

The configuration must specify:

- Each Kafka cluster
- Connection information for each cluster, including TLS authentication
- The replication flow and direction
  - Cluster to cluster
  - Topic to topic
- Replication rules
- Committed offset tracking intervals

This procedure describes how to implement MirrorMaker 2.0 by creating the configuration in a properties file, then passing the properties when using the MirrorMaker script file to set up the connections.



## NOTE

MirrorMaker 2.0 uses Kafka Connect to make the connections to transfer data between clusters. Kafka provides MirrorMaker sink and source connectors for data replication. If you wish to use the connectors instead of the MirrorMaker script, the connectors must be configured in the Kafka Connect cluster. For more information, refer to the [Apache Kafka documentation](#).

## Before you begin

A sample configuration properties file is provided in `./config/connect-mirror-maker.properties`.

## Prerequisites

- You need AMQ Streams installed on the hosts of each Kafka cluster node you are replicating.

## Procedure

- Open the sample properties file in a text editor, or create a new one, and edit the file to include connection information and the replication flows for each Kafka cluster.

The following example shows a configuration to connect two clusters, *cluster-1* and *cluster-2*, bidirectionally. Cluster names are configurable through the **clusters** property.

```
clusters=cluster-1,cluster-2 1

cluster-1.bootstrap.servers=<my-cluster>kafka-bootstrap-<my-project>:443 2
cluster-1.security.protocol=SSL 3
cluster-1.ssl.truststore.password=<my-truststore-password>
cluster-1.ssl.truststore.location=<path-to-truststore>/truststore.cluster-1.jks
cluster-1.ssl.keystore.password=<my-keystore-password>
cluster-1.ssl.keystore.location=<path-to-keystore>/user.cluster-1.p12

cluster-2.bootstrap.servers=<my-cluster>kafka-bootstrap-<my-project>:443 4
cluster-2.security.protocol=SSL 5
cluster-2.ssl.truststore.password=<my-truststore-password>
cluster-2.ssl.truststore.location=<path-to-truststore>/truststore.cluster-2.jks
cluster-2.ssl.keystore.password=<my-keystore-password>
cluster-2.ssl.keystore.location=<path-to-keystore>/user.cluster-2.p12

cluster-1->cluster-2.enabled=true 6
cluster-1->cluster-2.topics=* 7
cluster-2->cluster-1.enabled=true 8
cluster-2->cluster-1B->C.topics=* 9

replication.policy.separator=- 10
sync.topic.acls.enabled=false 11
refresh.topics.interval.seconds=60 12
refresh.groups.interval.seconds=60 13
```

- Each Kafka cluster is identified with its alias
- Connection information for *cluster-1*, using the *bootstrap* address and port 443. Both clusters use port 443 to connect to Kafka using OpenShift *Routes*

- 3 The **ssl.** properties define TLS configuration for *cluster-1*.
  - 4 Connection information for *cluster-2*.
  - 5 The **ssl.** properties define the TLS configuration for *cluster-2*.
  - 6 Replication flow enabled from the *cluster-1* cluster to the *cluster-2* cluster
  - 7 Replicates all topics from the *cluster-1* cluster to the *cluster-2* cluster
  - 8 Replication flow enabled from the *cluster-2* cluster to the *cluster-1* cluster
  - 9 Replicates specific topics from the *cluster-2* cluster to the *cluster-1* cluster
  - 10 Policy to declare the reserved character used as the topic name separator.
  - 11 When enabled, ACLs are applied to synchronized topics. The default is **false**.
  - 12 The period between checks for new topics to synchronize.
  - 13 The period between checks for new consumer groups to synchronize.
2. Start MirrorMaker with the cluster connection configuration and replication policies you defined in your properties file:

```
su - kafka
/opt/kafka/bin/kafka-mirror-maker.sh /config/connect-mirror-maker.properties
```

MirrorMaker sets up connections between the clusters.

3. Start ZooKeeper and Kafka in the target clusters:

```
su - kafka
/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties
```

```
su - kafka
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
```

4. For each target cluster, verify that the topics are being replicated:

```
/bin/kafka-topics.sh --zookeeper <ZooKeeperAddress> --list
```

## 9.5. USING MIRRORMAKER 2.0 IN LEGACY MODE

This procedure describes how to configure MirrorMaker 2.0 to use it in legacy mode. Legacy mode supports the previous version of MirrorMaker.

The MirrorMaker script `/opt/kafka/bin/kafka-mirror-maker.sh` can run MirrorMaker 2.0 in legacy mode.

### Prerequisites

You need the properties files you currently use with the legacy version of MirrorMaker.

- `/opt/kafka/config/consumer.properties`

- `/opt/kafka/config/producer.properties`

## Procedure

1. Edit the MirrorMaker **consumer.properties** and **producer.properties** files to turn off MirrorMaker 2.0 features.

For example:

```
replication.policy.class=org.apache.kafka.mirror.LegacyReplicationPolicy 1
refresh.topics.enabled=false 2
refresh.groups.enabled=false
emit.checkpoints.enabled=false
emit.heartbeats.enabled=false
sync.topic.configs.enabled=false
sync.topic.acls.enabled=false
```

- 1 Emulate the previous version of MirrorMaker.
- 2 MirrorMaker 2.0 features disabled, including the internal *checkpoint* and *heartbeat* topics

2. Save the changes and restart MirrorMaker with the properties files you used with the previous version of MirrorMaker:

```
su - kafka /opt/kafka/bin/kafka-mirror-maker.sh \
--consumer.config /opt/kafka/config/consumer.properties \
--producer.config /opt/kafka/config/producer.properties \
--num.streams=2
```

The **consumer** properties provide the configuration for the source cluster and the **producer** properties provide the target cluster configuration.

MirrorMaker sets up connections between the clusters.

3. Start ZooKeeper and Kafka in the target cluster:

```
su - kafka
/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties
```

```
su - kafka
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
```

4. For the target cluster, verify that the topics are being replicated:

```
/bin/kafka-topics.sh --zookeeper <ZooKeeperAddress> --list
```

## CHAPTER 10. KAFKA CLIENTS

The **kafka-clients** JAR file contains the Kafka Producer and Consumer APIs together with the Kafka AdminClient API.

- The Producer API allows applications to send data to a Kafka broker.
- The Consumer API allows applications to consume data from a Kafka broker.
- The AdminClient API provides functionality for managing Kafka clusters, including topics, brokers, and other components.

### 10.1. ADDING KAFKA CLIENTS AS A DEPENDENCY TO YOUR MAVEN PROJECT

This procedure shows you how to add the AMQ Streams Java clients as a dependency to your Maven project.

#### Prerequisites

- A Maven project with an existing **pom.xml**.

#### Procedure

1. Add the Red Hat Maven repository to the **<repositories>** section of your **pom.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <!-- ... -->

  <repositories>
    <repository>
      <id>redhat-maven</id>
      <url>https://maven.repository.redhat.com/ga/</url>
    </repository>
  </repositories>

  <!-- ... -->

</project>
```

2. Add the clients to the **<dependencies>** section of your **pom.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <!-- ... -->
```

```
<dependencies>
  <dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.4.0.redhat-00005</version>
  </dependency>
</dependencies>

<!-- ... -->
</project>
```

3. Build your Maven project.

## CHAPTER 11. KAFKA STREAMS API OVERVIEW

The Kafka Streams API allows applications to receive data from one or more input streams, execute complex operations like mapping, filtering or joining, and write the results into one or more output streams. It is part of the **kafka-streams** JAR package that is available in the Red Hat Maven repository.

### 11.1. ADDING THE KAFKA STREAMS API AS A DEPENDENCY TO YOUR MAVEN PROJECT

This procedure shows you how to add the AMQ Streams Java clients as a dependency to your Maven project.

#### Prerequisites

- A Maven project with an existing **pom.xml**.

#### Procedure

1. Add the Red Hat Maven repository to the **<repositories>** section of your **pom.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <!-- ... -->

  <repositories>
    <repository>
      <id>redhat-maven</id>
      <url>https://maven.repository.redhat.com/ga/</url>
    </repository>
  </repositories>

  <!-- ... -->

</project>
```

2. Add **kafka-streams** to the **<dependencies>** section of your **pom.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <!-- ... -->

  <dependencies>
    <dependency>
      <groupId>org.apache.kafka</groupId>
      <artifactId>kafka-streams</artifactId>
      <version>2.4.0.redhat-00005</version>
```

```
</dependency>  
</dependencies>  
  
<!-- ... -->  
</project>
```

3. Build your Maven project.



## CHAPTER 12. KAFKA BRIDGE

This chapter provides an overview of the AMQ Streams Kafka Bridge on Red Hat Enterprise Linux and helps you get started using its REST API to interact with AMQ Streams. To try out the Kafka Bridge in your local environment, see the [Section 12.2, “Kafka Bridge quickstart”](#) later in this chapter.

### Additional resources

- To view the API documentation, including example requests and responses, see the [Kafka Bridge API reference](#)[Kafka Bridge API reference^].
- To configure the Kafka Bridge for distributed tracing, see [Section 13.4, “Enabling tracing for the Kafka Bridge”](#).

## 12.1. KAFKA BRIDGE OVERVIEW

The AMQ Streams Kafka Bridge provides an API for integrating HTTP-based clients with a Kafka cluster running on Red Hat Enterprise Linux. The API enables such clients to produce and consume messages without the requirement to use the native Kafka protocol.

The API has two main resources – **consumers** and **topics** – that are exposed and made accessible through endpoints to interact with consumers and producers in your Kafka cluster. The resources relate only to the Kafka Bridge, not the consumers and producers connected directly to Kafka.

You can:

- Send messages to a topic.
- Create and delete consumers.
- Subscribe consumers to topics, so that they start receiving messages from those topics.
- Retrieve a list of the topics to which a consumer is subscribed.
- Unsubscribe consumers from topics.
- Assign partitions to consumers.
- Retrieve messages from topics.
- Commit a list of consumer offsets.
- Seek on a partition, so that a consumer starts receiving messages from the first or last offset position, or a given offset position.

Similar to an AMQ Streams installation, you can download the Kafka Bridge files for installation on Red Hat Enterprise Linux. See [Section 12.1.4, “Downloading a Kafka Bridge archive”](#) .

For more information on configuring the host and port for the **KafkaBridge** resource, see [Section 12.1.5, “Configuring Kafka Bridge properties”](#).

### 12.1.1. Requests to the Kafka Bridge

#### 12.1.1.1. Authentication and encryption

Authentication and encryption between HTTP clients and the Kafka Bridge is not yet supported. This means that requests sent from clients to the Kafka Bridge are:

- Not encrypted, and must use HTTP rather than HTTPS
- Sent without authentication

You can configure [TLS or SASL-based authentication](#) between the Kafka Bridge and your Kafka cluster.

You configure the Kafka Bridge for authentication through its [properties file](#).

### 12.1.1.2. Data formats and headers

Specify data formats and HTTP headers to ensure valid requests are submitted to the Kafka Bridge.

API request and response bodies are always encoded as JSON.

#### 12.1.1.2.1. Content Type headers

A **Content-Type** header must be submitted for all requests (apart from the exception described below).

Consumer operations (`/consumers` endpoints) and producer operations (`/topics` endpoints) require different **Content-Type** headers.

#### Content-Type headers for consumer operations

Regardless of the [embedded data format](#), **POST** requests for consumer operations must provide the following **Content-Type** header if the request body contains data:

**Content-Type:** `application/vnd.kafka.v2+json`

Do not send a **Content-Type** header if the **POST** request body is empty, or the request will fail.

#### Content-Type headers for producer operations

When performing producer operations, **POST** requests must provide the following **Content-Type** header specifying the desired *embedded data format*, either **json** or **binary**, as shown in the following table.

Embedded data format	Content-Type header
JSON	<b>Content-Type:</b> <code>application/vnd.kafka.json.v2+json</code>
Binary	<b>Content-Type:</b> <code>application/vnd.kafka.binary.v2+json</code>

You set the embedded data format when creating a consumer using the `consumers/groupid` endpoint –for more information, see the next section.

The **Content-Type** must not be set if the **POST** request has an empty body. An empty body can be used to create a consumer with the default values.

#### 12.1.1.2.2. Embedded data format

The embedded data format is the format of the Kafka messages that are transmitted, over HTTP, from a producer to a consumer using the Kafka Bridge. Two embedded data formats are supported: JSON or binary.

When creating a consumer using the `/consumers/groupid` endpoint, the **POST** request body must specify an embedded data format of either JSON or binary. This is specified in the **format** field in the request body, for example:

```
{
  "name": "my-consumer",
  "format": "binary", 1
  ...
}
```

**1** A binary embedded data format.

If an embedded data format for the consumer is not specified, then a binary format is set.

The embedded data format specified when creating a consumer must match the data format of the Kafka messages it will consume.

If you choose to specify a binary embedded data format, subsequent producer requests must provide the binary data in the request body as Base64-encoded strings. For example, when sending messages by making **POST** requests to the `/topics/topicname` endpoint, the **value** must be encoded in Base64:

```
{
  "records": [
    {
      "key": "my-key",
      "value": "ZWR3YXJkdGhldGhyZWVsZWdnZWVjYXQ="
    },
  ]
}
```

Producer requests must also provide a **Content-Type** header that corresponds to the embedded data format, for example, **Content-Type: application/vnd.kafka.binary.v2+json**.

### 12.1.1.2.3. Accept headers

After creating a consumer, all subsequent GET requests must provide an **Accept** header in the following format:

```
Accept: application/vnd.kafka.embedded-data-format.v2+json
```

The *embedded-data-format* is either **json** or **binary**.

For example, when retrieving records for a subscribed consumer using an embedded data format of JSON, include this Accept header:

```
Accept: application/vnd.kafka.json.v2+json
```

## 12.1.2. Configuring loggers for the Kafka Bridge

The AMQ Streams Kafka bridge allows you to set a different log level for each operation that is defined by the related OpenAPI specification.

Each operation has a corresponding API endpoint through which the bridge receives requests from HTTP clients. You can change the log level on each endpoint to produce more or less fine-grained logging information about the incoming and outgoing HTTP requests.

Loggers are defined in the **log4j.properties** file, which has the following default configuration for **healthy** and **ready** endpoints:

```
log4j.logger.http.openapi.operation.healthy=WARN, out
log4j.additivity.http.openapi.operation.healthy=false
log4j.logger.http.openapi.operation.ready=WARN, out
log4j.additivity.http.openapi.operation.ready=false
```

The log level of all other operations is set to **INFO** by default. Loggers are formatted as follows:

```
log4j.logger.http.openapi.operation.<operation-id>
```

Where **<operation-id>** is the identifier of the specific operation. Following is the list of operations defined by the OpenAPI specification:

- **createConsumer**
- **deleteConsumer**
- **subscribe**
- **unsubscribe**
- **poll**
- **assign**
- **commit**
- **send**
- **sendToPartition**
- **seekToBeginning**
- **seekToEnd**
- **seek**
- **healthy**
- **ready**
- **openapi**

### 12.1.3. Kafka Bridge API resources

For the full list of REST API endpoints and descriptions, including example requests and responses, see the [Kafka Bridge API reference](#) [Kafka Bridge API reference^].

## 12.1.4. Downloading a Kafka Bridge archive

A zipped distribution of the AMQ Streams Kafka Bridge is available for download from the Red Hat website.

### Procedure

- Download the latest version of the Red Hat AMQ Streams Kafka Bridge archive from the [Customer Portal](#).

## 12.1.5. Configuring Kafka Bridge properties

This procedure describes how to configure the properties used by the AMQ Streams Kafka Bridge.

You configure the Kafka Bridge, as any other Kafka client, using appropriate prefixes for Kafka-related properties.

- **kafka.** for general configuration that applies to producers and consumers, such as server connection and security.
- **kafka.consumer.** for consumer-specific configuration passed only to the consumer.
- **kafka.producer.** for producer-specific configuration passed only to the producer.

### Prerequisites

- [AMQ Streams is installed on the host](#)
- [The Kafka Bridge installation archive is downloaded](#)

### Procedure

1. Edit the **application.properties** file provided with the AMQ Streams Kafka Bridge installation archive.

Use the properties file to specify Kafka and HTTP-related properties, and to enable and disable distributed tracing.

- a. Configure standard Kafka-related properties, including properties specific to the Kafka consumers and producers.

Use:

- **kafka.bootstrap.servers** to define the host/port connections to the Kafka cluster
- **kafka.producer.acks** to provide acknowledgments to the HTTP client
- **kafka.consumer.auto.offset.reset** to determine how to manage reset of the offset in Kafka

For more information on configuration of Kafka properties, see the [Apache Kafka website](#)

- b. Configure HTTP-related properties to enable HTTP access to the Kafka cluster.

```
http.enabled=true
http.host=0.0.0.0
http.port=8080
```

- c. Enable and disable distributed tracing.

```
bridge.tracing=jaeger
```

### Additional resources

- [Chapter 13, \*Distributed tracing\*](#)
- [Section 13.4, “Enabling tracing for the Kafka Bridge”](#)

## 12.1.6. Installing the Kafka Bridge

Follow this procedure to install the AMQ Streams Kafka Bridge on Red Hat Enterprise Linux.

### Prerequisites

- [AMQ Streams is installed on the host](#)
- [The Kafka Bridge installation archive is downloaded](#)
- [The Kafka Bridge configuration properties are set](#)

### Procedure

1. If you have not already done so, unzip the AMQ Streams Kafka Bridge installation archive to any directory.
2. Run the Kafka Bridge script using the configuration properties as a parameter:  
For example:

```
./bin/kafka_bridge_run.sh --config-file=_path_/configfile.properties
```

3. Check to see that the installation was successful in the log.

```
HTTP-Kafka Bridge started and listening on port 8080  
HTTP-Kafka Bridge bootstrap servers localhost:9092
```

## 12.2. KAFKA BRIDGE QUICKSTART

Use this quickstart to try out the AMQ Streams Kafka Bridge on Red Hat Enterprise Linux. You will learn how to:

- Install the Kafka Bridge
- Produce messages to topics and partitions in your Kafka cluster
- Create a Kafka Bridge consumer
- Perform basic consumer operations, such as subscribing the consumer to topics and retrieving the messages that you produced

In this quickstart, HTTP requests are formatted as curl commands that you can copy and paste to your terminal.

Ensure you have the prerequisites and then follow the tasks in the order provided in this chapter.

## About data formats

In this quickstart, you will produce and consume messages in JSON format, not binary. For more information on the data formats and HTTP headers used in the example requests, see [Section 12.1.1, “Requests to the Kafka Bridge”](#).

## Prerequisites for the quickstart

- [AMQ Streams is installed on the host](#)
- [A single node AMQ Streams cluster is running](#)
- [The Kafka Bridge installation archive is downloaded](#)

### 12.2.1. Deploying the Kafka Bridge locally

Deploy an instance of the AMQ Streams Kafka Bridge to the host. Use the **application.properties** file provided with the installation archive to apply the default configuration settings.

#### Procedure

1. Open the **application.properties** file and check that the default **HTTP related settings** are defined:

```
http.enabled=true
http.host=0.0.0.0
http.port=8080
```

This configures the Kafka Bridge to listen for requests on port 8080.

2. Run the Kafka Bridge script using the configuration properties as a parameter:

```
./bin/kafka_bridge_run.sh --config-file=<path>/application.properties
```

#### What to do next

- [Produce messages to topics and partitions](#) .

### 12.2.2. Producing messages to topics and partitions

Produce messages to a topic in JSON format by using the [topics](#) endpoint.

You can specify destination partitions for messages in the request body, as shown below. The [partitions](#) endpoint provides an alternative method for specifying a single destination partition for all messages as a path parameter.

#### Procedure

1. Create a Kafka topic using the **kafka-topics.sh** utility:

```
bin/kafka-topics.sh --zookeeper localhost:2181 --create --topic bridge-quickstart-topic --
partitions 3 --replication-factor 1 --config retention.ms=7200000 --config
segment.bytes=1073741824
```

Specify three partitions.

2. Verify that the topic was created:

```
bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic bridge-quickstart-topic
```

3. Using the Kafka Bridge, produce three messages to the topic you created:

```
curl -X POST \
  http://localhost:8080/topics/bridge-quickstart-topic \
  -H 'content-type: application/vnd.kafka.json.v2+json' \
  -d '{
    "records": [
      {
        "key": "my-key",
        "value": "sales-lead-0001"
      },
      {
        "value": "sales-lead-0002",
        "partition": 2
      },
      {
        "value": "sales-lead-0003"
      }
    ]
  }'
```

- **sales-lead-0001** is sent to a partition based on the hash of the key.
  - **sales-lead-0002** is sent directly to partition 2.
  - **sales-lead-0003** is sent to a partition in the **bridge-quickstart-topic** topic using a round-robin method.
4. If the request is successful, the Kafka Bridge returns an **offsets** array, along with a **200** (OK) code and a **content-type** header of **application/vnd.kafka.v2+json**. For each message, the **offsets** array describes:
    - The partition that the message was sent to
    - The current message offset of the partition

#### Example response

```
#...
{
  "offsets":[
    {
      "partition":0,
      "offset":0
    },
  ],
}
```



```
{
  "partition":2,
  "offset":0
},
{
  "partition":0,
  "offset":1
}
]
```

### What to do next

After producing messages to topics and partitions, [create a Kafka Bridge consumer](#).

### Additional resources

- [POST /topics/{topicname}](#) in the API reference documentation.
- [POST /topics/{topicname}/partitions/{partitionid}](#) in the API reference documentation.

### 12.2.3. Creating a Kafka Bridge consumer

Before you can perform any consumer operations on the Kafka cluster, you must first create a consumer by using the [consumers](#) endpoint. The consumer is referred to as a *Kafka Bridge consumer*.

#### Procedure

1. Create a Kafka Bridge consumer in a new consumer group named **bridge-quickstart-consumer-group**:

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group \
-H 'content-type: application/vnd.kafka.v2+json' \
-d '{
  "name": "bridge-quickstart-consumer",
  "auto.offset.reset": "earliest",
  "format": "json",
  "enable.auto.commit": false,
  "fetch.min.bytes": 512,
  "consumer.request.timeout.ms": 30000
}'
```

- The consumer is named **bridge-quickstart-consumer** and the embedded data format is set as **json**.
- The consumer will not commit offsets to the log automatically because the **enable.auto.commit** setting is **false**. You will commit the offsets manually later in this quickstart.



#### NOTE

The Kafka Bridge generates a random consumer name if you do not specify a consumer name in the request body.

If the request is successful, the Kafka Bridge returns the consumer ID (**instance\_id**) and base URL (**base\_uri**) in the response body, along with a **200** (OK) code.

### Example response

```
#...
{
  "instance_id": "bridge-quickstart-consumer",
  "base_uri": "http://<bridge-name>-bridge-service:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer"
}
```

2. Copy the base URL (**base\_uri**) to use in the other consumer operations in this quickstart.

### What to do next

Now that you have created a Kafka Bridge consumer, you can [subscribe it to topics](#).

### Additional resources

- [POST /consumers/{groupid}](#) in the API reference documentation.

## 12.2.4. Subscribing a Kafka Bridge consumer to topics

Subscribe the Kafka Bridge consumer to one or more topics by using the [subscription](#) endpoint. Once subscribed, the consumer starts receiving all messages that are produced to the topic.

### Procedure

- Subscribe the consumer to the **bridge-quickstart-topic** topic that you created earlier, in [Producing messages to topics and partitions](#):

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/subscription \
-H 'content-type: application/vnd.kafka.v2+json' \
-d '{
  "topics": [
    "bridge-quickstart-topic"
  ]
}'
```

The **topics** array can contain a single topic (as shown above) or multiple topics. If you want to subscribe the consumer to multiple topics that match a regular expression, you can use the **topic\_pattern** string instead of the **topics** array.

If the request is successful, the Kafka Bridge returns a **204 No Content** code only.

### What to do next

After subscribing a Kafka Bridge consumer to topics, you can [retrieve messages from the consumer](#).

### Additional resources

- [POST /consumers/{groupid}/instances/{name}/subscription](#) in the API reference documentation.

## 12.2.5. Retrieving the latest messages from a Kafka Bridge consumer

Retrieve the latest messages from the Kafka Bridge consumer by requesting data from the [records](#) endpoint. In production, HTTP clients can call this endpoint repeatedly (in a loop).

### Procedure

1. Produce additional messages to the Kafka Bridge consumer, as described in [Producing messages to topics and partitions](#).
2. Submit a **GET** request to the **records** endpoint:

```
curl -X GET http://localhost:8080/consumers/bridge-quickstart-consumer-
group/instances/bridge-quickstart-consumer/records \
-H 'accept: application/vnd.kafka.json.v2+json'
```

After creating and subscribing to a Kafka Bridge consumer, a first GET request will return an empty response because the poll operation triggers a rebalancing process to assign partitions.

3. Repeat step two to retrieve messages from the Kafka Bridge consumer. The Kafka Bridge returns an array of messages – describing the topic name, key, value, partition, and offset – in the response body, along with a **200** (OK) code. Messages are retrieved from the latest offset by default.

```
HTTP/1.1 200 OK
content-type: application/vnd.kafka.json.v2+json
#...
[
  {
    "topic":"bridge-quickstart-topic",
    "key":"my-key",
    "value":"sales-lead-0001",
    "partition":0,
    "offset":0
  },
  {
    "topic":"bridge-quickstart-topic",
    "key":null,
    "value":"sales-lead-0003",
    "partition":0,
    "offset":1
  },
  #...
```



### NOTE

If an empty response is returned, produce more records to the consumer as described in [Producing messages to topics and partitions](#), and then try retrieving messages again.

### What to do next

After retrieving messages from a Kafka Bridge consumer, try [committing offsets to the log](#).

### Additional resources

- [GET /consumers/{groupid}/instances/{name}/records](#) in the API reference documentation.

## 12.2.6. Committing offsets to the log

Use the [offsets](#) endpoint to manually commit offsets to the log for all messages received by the Kafka Bridge consumer. This is required because the Kafka Bridge consumer that you created earlier, in [Creating a Kafka Bridge consumer](#), was configured with the **enable.auto.commit** setting as **false**.

### Procedure

- Commit offsets to the log for the **bridge-quickstart-consumer**:

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/offsets
```

Because no request body is submitted, offsets are committed for all the records that have been received by the consumer. Alternatively, the request body can contain an array ([OffsetCommitSeekList](#)) that specifies the topics and partitions that you want to commit offsets for.

If the request is successful, the Kafka Bridge returns a **204 No Content** code only.

### What to do next

After committing offsets to the log, try out the endpoints for [seeking to offsets](#).

### Additional resources

- [POST /consumers/{groupid}/instances/{name}/offsets](#) in the API reference documentation.

## 12.2.7. Seeking to offsets for a partition

Use the [positions](#) endpoints to configure the Kafka Bridge consumer to retrieve messages for a partition from a specific offset, and then from the latest offset. This is referred to in Apache Kafka as a seek operation.

### Procedure

1. Seek to a specific offset for partition 0 of the **quickstart-bridge-topic** topic:

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-group/instances/bridge-quickstart-consumer/positions \
-H 'content-type: application/vnd.kafka.v2+json' \
-d '{
  "offsets": [
    {
      "topic": "bridge-quickstart-topic",
      "partition": 0,
      "offset": 2
    }
  ]
}'
```

If the request is successful, the Kafka Bridge returns a **204 No Content** code only.

2. Submit a **GET** request to the **records** endpoint:

```
curl -X GET http://localhost:8080/consumers/bridge-quickstart-consumer-
group/instances/bridge-quickstart-consumer/records \
-H 'accept: application/vnd.kafka.json.v2+json'
```

The Kafka Bridge returns messages from the offset that you sought to.

3. Restore the default message retrieval behavior by seeking to the last offset for the same partition. This time, use the [positions/end](#) endpoint.

```
curl -X POST http://localhost:8080/consumers/bridge-quickstart-consumer-
group/instances/bridge-quickstart-consumer/positions/end \
-H 'content-type: application/vnd.kafka.v2+json' \
-d '{
  "partitions": [
    {
      "topic": "bridge-quickstart-topic",
      "partition": 0
    }
  ]
}'
```

If the request is successful, the Kafka Bridge returns another **204 No Content** code.



#### NOTE

You can also use the [positions/beginning](#) endpoint to seek to the first offset for one or more partitions.

### What to do next

In this quickstart, you have used the AMQ Streams Kafka Bridge to perform several common operations on a Kafka cluster. You can now [delete the Kafka Bridge consumer](#) that you created earlier.

### Additional resources

- [POST /consumers/{groupid}/instances/{name}/positions](#) in the API reference documentation.
- [POST /consumers/{groupid}/instances/{name}/positions/beginning](#) in the API reference documentation.
- [POST /consumers/{groupid}/instances/{name}/positions/end](#) in the API reference documentation.

## 12.2.8. Deleting a Kafka Bridge consumer

Finally, delete the Kafka Bridge consumer that you used throughout this quickstart.

### Procedure

- Delete the Kafka Bridge consumer by sending a **DELETE** request to the [instances](#) endpoint.

```
curl -X DELETE http://localhost:8080/consumers/bridge-quickstart-consumer-  
group/instances/bridge-quickstart-consumer
```

If the request is successful, the Kafka Bridge returns a **204 No Content** code only.

### Additional resources

- [DELETE /consumers/{groupid}/instances/{name}](#) in the API reference documentation.

## CHAPTER 13. DISTRIBUTED TRACING

AMQ Streams on Red Hat Enterprise Linux supports distributed tracing with Jaeger for:

- Kafka Producers, Kafka Consumers, and Kafka Streams applications (referred to as *Kafka clients*)
- MirrorMaker and Kafka Connect
- Kafka Bridge

The steps to configure distributed tracing vary by client and component. However, three high-level tasks are involved:

1. Enable a Jaeger tracer for the client or component.
2. Enable the Interceptors for the client or component.
  - For Kafka clients, enabling the Interceptors involves *instrumenting* your application code for OpenTracing.
  - For MirrorMaker, Kafka Connect, and the Kafka Bridge, enabling the Interceptors involves setting configuration properties for each component.
3. Set tracing environment variables before deploying the client or component.

This chapter provides an overview of distributed tracing, as well as instructions for configuring supported clients and components. Setting up distributed tracing for applications and systems beyond AMQ Streams is outside the scope of this chapter. To learn more about this subject, see the [OpenTracing documentation](#) and search for "inject and extract".



### NOTE

Distributed tracing is not supported for Kafka brokers.

### Prerequisites

- The Jaeger backend components are deployed to the host operating system. For deployment instructions, see the [Jaeger deployment documentation](#).

## 13.1. OVERVIEW OF DISTRIBUTED TRACING

Distributed tracing allows developers and system administrators to track the progress of transactions between applications (and services in a microservice architecture) in a distributed system. This information is useful for monitoring application performance and investigating issues with target systems and end-user applications.

In AMQ Streams, distributed tracing facilitates the end-to-end tracking of messages: from source systems to the Kafka cluster and then to target systems and applications.

As an aspect of system observability, distributed tracing complements the metrics that are available to view in [Grafana](#) dashboards and the available loggers for each component.

### 13.1.1. OpenTracing and Jaeger

The [OpenTracing](#) and [Jaeger](#) projects are used to implement distributed tracing in AMQ Streams.

## OpenTracing

The OpenTracing specification defines APIs that developers can use to instrument applications for distributed tracing. When you instrument an application, you add instrumentation code in order to monitor the execution of individual transactions. When instrumented, applications generate *traces* when individual transactions occur. Traces are composed of *spans*, which define specific units of work.

To simplify the instrumentation of Kafka clients, AMQ Streams includes the [OpenTracing Apache Kafka Client Instrumentation](#) library.



### NOTE

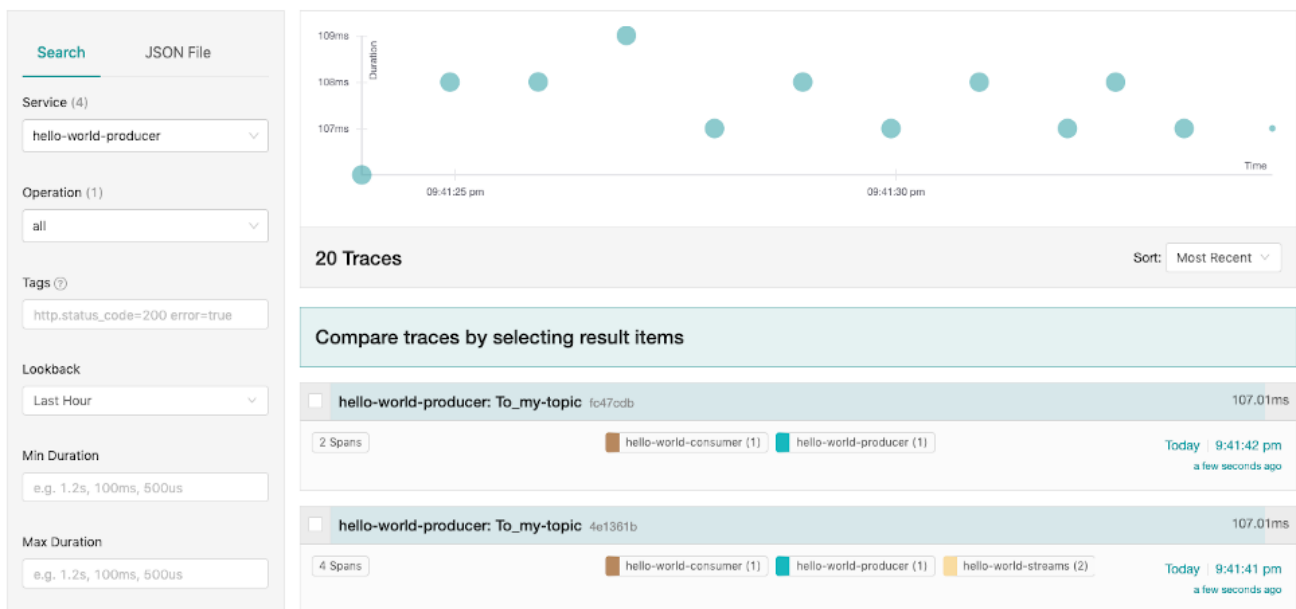
The OpenTracing project is merging with the OpenCensus project to form the [OpenTelemetry](#) project. OpenTelemetry will provide compatibility for applications that are instrumented using the OpenTracing APIs.

## Jaeger

Jaeger, a tracing system, is an implementation of the OpenTracing APIs. Jaeger is used for monitoring and troubleshooting microservices-based distributed systems and provides client libraries for instrumenting applications.

Jaeger samples the total traces generated by an application, based on a set sampling strategy, and then visualizes them in a user interface. This allows you to visualize, query, filter, and analyze trace data.

### An example of a query in the Jaeger user interface



## 13.2. CONFIGURING KAFKA CLIENTS FOR TRACING

This section describes how to configure Kafka clients (Kafka Producers, Kafka Consumers, and Kafka Streams applications) for distributed tracing.

### 13.2.1. Enabling a Jaeger tracer for Kafka clients

Configure and enable a Jaeger tracer using the [tracing environment variables](#).



## Procedure

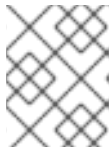
Perform the following steps for each Kafka client (Kafka Producer, Kafka Consumer, and Kafka Streams application):

1. Add Maven dependencies for Jaeger to the **pom.xml** file for the client:

```
<dependency>
  <groupId>io.jaegertracing</groupId>
  <artifactId>jaeger-client</artifactId>
  <version>1.0.0</version>
</dependency>
```

2. Define the configuration of the Jaeger tracer using the [tracing environment variables](#).
3. Create the Jaeger tracer from the environment variables that you defined in step two:

```
Tracer tracer = Configuration.fromEnv().getTracer();
```



### NOTE

For alternative ways to initialize a Jaeger tracer, see the [Java OpenTracing library](#) documentation.

4. Register the Jaeger tracer as a global tracer:

```
GlobalTracer.register(tracer);
```

A Jaeger tracer is now enabled for the Kafka client to use.

## 13.2.2. Instrumenting Kafka Producers and Consumers for tracing

Use a Decorator pattern or Interceptors to instrument your Kafka Producer and Consumer application code for distributed tracing. When instrumented, the Interceptors in the Kafka Producer or Consumer are enabled.

### Procedure

Perform the following steps in the application code of each Kafka Producer and Consumer:

1. Add a Maven dependency for OpenTracing to the Producer or Consumer's **pom.xml** file.

```
<dependency>
  <groupId>io.opentracing.contrib</groupId>
  <artifactId>opentracing-kafka-client</artifactId>
  <version>0.1.4</version>
</dependency>
```

2. Instrument your client application code using either a Decorator pattern or Interceptors.
  - If you prefer to use a Decorator pattern, use following example:

```
// Create an instance of the KafkaProducer:
KafkaProducer<Integer, String> producer = new KafkaProducer<>(senderProps);
```

```

// Create an instance of the TracingKafkaProducer:
TracingKafkaProducer<Integer, String> tracingProducer = new TracingKafkaProducer<>
(producer,
    tracer);

// Send:
tracingProducer.send(...);

// Create an instance of the KafkaConsumer:
KafkaConsumer<Integer, String> consumer = new KafkaConsumer<>(consumerProps);

// Create an instance of the TracingKafkaConsumer:
TracingKafkaConsumer<Integer, String> tracingConsumer = new
TracingKafkaConsumer<>(consumer,
    tracer);

// Subscribe:
tracingConsumer.subscribe(Collections.singletonList("messages"));

// Get messages:
ConsumerRecords<Integer, String> records = tracingConsumer.poll(1000);

// Retrieve SpanContext from polled record (consumer side):
ConsumerRecord<Integer, String> record = ...
SpanContext spanContext = TracingKafkaUtils.extractSpanContext(record.headers(),
tracer);

```

- If you prefer to use Interceptors, use the following example:

```

// Register the tracer with GlobalTracer:
GlobalTracer.register(tracer);

// Add the TracingProducerInterceptor to the sender properties:
senderProps.put(ProducerConfig.INTERCEPTOR_CLASSES_CONFIG,
    TracingProducerInterceptor.class.getName());

// Create an instance of the KafkaProducer:
KafkaProducer<Integer, String> producer = new KafkaProducer<>(senderProps);

// Send:
producer.send(...);

// Add the TracingConsumerInterceptor to the consumer properties:
consumerProps.put(ConsumerConfig.INTERCEPTOR_CLASSES_CONFIG,
    TracingConsumerInterceptor.class.getName());

// Create an instance of the KafkaConsumer:
KafkaConsumer<Integer, String> consumer = new KafkaConsumer<>(consumerProps);

// Subscribe:
consumer.subscribe(Collections.singletonList("messages"));

// Get messages:
ConsumerRecords<Integer, String> records = consumer.poll(1000);

```

```

// Retrieve the SpanContext from a polled message (consumer side):
ConsumerRecord<Integer, String> record = ...
SpanContext spanContext = TracingKafkaUtils.extractSpanContext(record.headers(),
tracer);

```

### 13.2.2.1. Custom span names in a Decorator pattern

A *span* is a logical unit of work in Jaeger, with an operation name, start time, and duration.

If you use a Decorator pattern to instrument your Kafka Producer and Consumer applications, you can define custom span names by passing a **BiFunction** object as an additional argument when creating the **TracingKafkaProducer** and **TracingKafkaConsumer** objects. The OpenTracing Apache Kafka Client Instrumentation library includes several built-in span names, which are described below.

#### Example: Using custom span names to instrument client application code in a Decorator pattern

```

// Create a BiFunction for the KafkaProducer that operates on (String operationName,
// ProducerRecord consumerRecord) and returns a String to be used as the name:

BiFunction<String, ProducerRecord, String> producerSpanNameProvider =
    (operationName, producerRecord) -> "CUSTOM_PRODUCER_NAME";

// Create an instance of the KafkaProducer:
KafkaProducer<Integer, String> producer = new KafkaProducer<>(senderProps);

// Create an instance of the TracingKafkaProducer
TracingKafkaProducer<Integer, String> tracingProducer = new TracingKafkaProducer<>(producer,
    tracer,
    producerSpanNameProvider);

// Spans created by the tracingProducer will now have "CUSTOM_PRODUCER_NAME" as the span
// name.

// Create a BiFunction for the KafkaConsumer that operates on (String operationName,
// ConsumerRecord consumerRecord) and returns a String to be used as the name:

BiFunction<String, ConsumerRecord, String> consumerSpanNameProvider =
    (operationName, consumerRecord) -> operationName.toUpperCase();

// Create an instance of the KafkaConsumer:
KafkaConsumer<Integer, String> consumer = new KafkaConsumer<>(consumerProps);

// Create an instance of the TracingKafkaConsumer, passing in the consumerSpanNameProvider
// BiFunction:

TracingKafkaConsumer<Integer, String> tracingConsumer = new TracingKafkaConsumer<>
    (consumer,
    tracer,
    consumerSpanNameProvider);

// Spans created by the tracingConsumer will have the operation name as the span name, in upper-
// case.
// "receive" -> "RECEIVE"

```

### 13.2.2.2. Built-in span names

When defining custom span names, you can use the following **BiFunctions** in the **ClientSpanNameProvider** class. If no **spanNameProvider** is specified, **CONSUMER\_OPERATION\_NAME** and **PRODUCER\_OPERATION\_NAME** are used.

BiFunction	Description
<b>CONSUMER_OPERATION_NAME, PRODUCER_OPERATION_NAME</b>	Returns the <b>operationName</b> as the span name: "receive" for Consumers and "send" for Producers.
<b>CONSUMER_PREFIXED_OPERATION_NAME (String prefix), PRODUCER_PREFIXED_OPERATION_NAME (String prefix)</b>	Returns a String concatenation of <b>prefix</b> and <b>operationName</b> .
<b>CONSUMER_TOPIC, PRODUCER_TOPIC</b>	Returns the name of the topic that the message was sent to or retrieved from in the format <b>(record.topic())</b> .
<b>PREFIXED_CONSUMER_TOPIC (String prefix), PREFIXED_PRODUCER_TOPIC (String prefix)</b>	Returns a String concatenation of <b>prefix</b> and the topic name in the format <b>(record.topic())</b> .
<b>CONSUMER_OPERATION_NAME_TOPIC, PRODUCER_OPERATION_NAME_TOPIC</b>	Returns the operation name and the topic name: <b>"operationName - record.topic()"</b> .
<b>CONSUMER_PREFIXED_OPERATION_NAME_TOPIC (String prefix), PRODUCER_PREFIXED_OPERATION_NAME_TOPIC (String prefix)</b>	Returns a String concatenation of <b>prefix</b> and <b>"operationName - record.topic()"</b> .

### 13.2.3. Instrumenting Kafka Streams applications for tracing

Instrument Kafka Streams applications for distributed tracing using a supplier interface. This enables the Interceptors in the application.

#### Procedure

Perform the following steps for each Kafka Streams application:

1. Add the **opentracing-kafka-streams** dependency to the Kafka Streams application's **pom.xml** file.

```
<dependency>
  <groupId>io.opentracing.contrib</groupId>
  <artifactId>opentracing-kafka-streams</artifactId>
  <version>0.1.4</version>
</dependency>
```

2. Create an instance of the **TracingKafkaClientSupplier** supplier interface:

```
KafkaClientSupplier supplier = new TracingKafkaClientSupplier(tracer);
```

3. Provide the supplier interface to **KafkaStreams**:

```
KafkaStreams streams = new KafkaStreams(builder.build(), new StreamsConfig(config),
supplier);
streams.start();
```

## 13.3. CONFIGURING MIRRORMAKER AND KAFKA CONNECT FOR TRACING

This section describes how to configure MirrorMaker and Kafka Connect for distributed tracing.

You must enable a Jaeger tracer for each component.

### 13.3.1. Enabling tracing for MirrorMaker

Enable distributed tracing for MirrorMaker by passing the Interceptor properties as consumer and producer configuration parameters.

Messages are traced from the source cluster to the target cluster; the trace data records messages entering and leaving the MirrorMaker component.

#### Procedure

1. Configure and enable a Jaeger tracer.
2. Edit the **/opt/kafka/config/consumer.properties** file.  
Add the following Interceptor property:

```
consumer.interceptor.classes=io.opentracing.contrib.kafka.TracingConsumerInterceptor
```

3. Edit the **/opt/kafka/config/producer.properties** file.  
Add the following Interceptor property:

```
producer.interceptor.classes=io.opentracing.contrib.kafka.TracingProducerInterceptor
```

4. Start MirrorMaker with the consumer and producer configuration files as parameters:

```
su - kafka
/opt/kafka/bin/kafka-mirror-maker.sh --consumer.config /opt/kafka/config/consumer.properties
--producer.config /opt/kafka/config/producer.properties --num.streams=2
```

### 13.3.2. Enabling tracing for Kafka Connect

Enable distributed tracing for Kafka Connect using configuration properties.

Only messages produced and consumed by Kafka Connect itself are traced. To trace messages sent between Kafka Connect and external systems, you must configure tracing in the connectors for those systems.

## Procedure

1. Configure and enable a Jaeger tracer.
2. Edit the relevant Kafka Connect configuration file.
  - If you are running Kafka Connect in standalone mode, edit the **/opt/kafka/config/connect-standalone.properties** file.
  - If you are running Kafka Connect in distributed mode, edit the **/opt/kafka/config/connect-distributed.properties** file.
3. Add the following properties to the configuration file:

```
producer.interceptor.classes=io.opentracing.contrib.kafka.TracingProducerInterceptor  
consumer.interceptor.classes=io.opentracing.contrib.kafka.TracingConsumerInterceptor
```

4. Save the configuration file.
5. Set tracing environment variables and then run Kafka Connect in standalone or distributed mode.

The Interceptors in Kafka Connect's internal consumers and producers are now enabled.

## Additional resources

- [Section 13.5, "Tracing environment variables"](#)
- [Section 8.1.3, "Running Kafka Connect in standalone mode"](#)
- [Section 8.2.3, "Running distributed Kafka Connect"](#)

## 13.4. ENABLING TRACING FOR THE KAFKA BRIDGE

Enable distributed tracing for the Kafka Bridge by editing the Kafka Bridge configuration file. You can then deploy a Kafka Bridge instance that is configured for distributed tracing to the host operating system.

Traces are generated for the following transactions:

- The Kafka Bridge sends messages to HTTP clients and consumes messages from HTTP clients.
- HTTP clients send HTTP requests to send and receive messages through the Kafka Bridge.

In order to have end-to-end tracing, you must also configure tracing in your HTTP clients.

## Procedure

1. Edit the **config/application.properties** file in the Kafka Bridge installation directory. Remove the code comments from the following line:

```
bridge.tracing=jaeger
```

2. Save the configuration file.

- Run the `bin/kafka_bridge_run.sh` script using the configuration properties as a parameter:

```
cd kafka-bridge-0.xy.x.redhat-0000x
./bin/kafka_bridge_run.sh --config-file=config/application.properties
```

The Interceptors in the Kafka Bridge's internal consumers and producers are now enabled.

#### Additional resources

- [Section 12.1.5, "Configuring Kafka Bridge properties"](#)

## 13.5. TRACING ENVIRONMENT VARIABLES

Use these environment variables when configuring a Jaeger tracer for Kafka clients and components.



### NOTE

The tracing environment variables are part of the Jaeger project and are subject to change. For the latest environment variables, see the [Jaeger documentation](#).

Property	Required	Description
<b>JAEGER_SERVICE_NAME</b>	Yes	The name of the Jaeger tracer service.
<b>JAEGER_AGENT_HOST</b>	No	The hostname for communicating with the <b>jaeger-agent</b> through the User Datagram Protocol (UDP).
<b>JAEGER_AGENT_PORT</b>	No	The port used for communicating with the <b>jaeger-agent</b> through UDP.
<b>JAEGER_ENDPOINT</b>	No	The traces endpoint. Only define this variable if the client application will bypass the <b>jaeger-agent</b> and connect directly to the <b>jaeger-collector</b> .
<b>JAEGER_AUTH_TOKEN</b>	No	The authentication token to send to the endpoint as a bearer token.
<b>JAEGER_USER</b>	No	The username to send to the endpoint if using basic authentication.
<b>JAEGER_PASSWORD</b>	No	The password to send to the endpoint if using basic authentication.

Property	Required	Description
<b>JAEGER_PROPAGATION</b>	No	A comma-separated list of formats to use for propagating the trace context. Defaults to the standard Jaeger format. Valid values are <b>jaeger</b> and <b>b3</b> .
<b>JAEGER_REPORTER_LOG_SPANS</b>	No	Indicates whether the reporter should also log the spans.
<b>JAEGER_REPORTER_MAX_QUEUE_SIZE</b>	No	The reporter's maximum queue size.
<b>JAEGER_REPORTER_FLUSH_INTERVAL</b>	No	The reporter's flush interval, in ms. Defines how frequently the Jaeger reporter flushes span batches.
<b>JAEGER_SAMPLER_TYPE</b>	No	<p>The sampling strategy to use for client traces: Constant, Probabilistic, Rate Limiting, or Remote (the default type).</p> <p>To sample all traces, use the Constant sampling strategy with a parameter of 1.</p> <p>For more information, see the <a href="#">Jaeger documentation</a>.</p>
<b>JAEGER_SAMPLER_PARAM</b>	No	The sampler parameter (number).
<b>JAEGER_SAMPLER_MANAGER_HOST_PORT</b>	No	The hostname and port to use if a Remote sampling strategy is selected.
<b>JAEGER_TAGS</b>	No	<p>A comma-separated list of tracer-level tags that are added to all reported spans.</p> <p>The value can also refer to an environment variable using the format <b>#{envVarName:default}</b>. <b>:default</b> is optional and identifies a value to use if the environment variable cannot be found.</p>



## CHAPTER 14. KAFKA EXPORTER

[Kafka Exporter](#) is an open source project to enhance monitoring of Apache Kafka brokers and clients.

Kafka Exporter is provided with AMQ Streams for deployment with a Kafka cluster to extract additional metrics data from Kafka brokers related to offsets, consumer groups, consumer lag, and topics.

The metrics data is used, for example, to help identify slow consumers.

Lag data is exposed as Prometheus metrics, which can then be presented in Grafana for analysis.

If you are already using Prometheus and Grafana for monitoring of built-in Kafka metrics, you can configure Prometheus to also scrape the Kafka Exporter Prometheus endpoint.

### Additional resources

Kafka exposes metrics through JMX, which can then be exported as Prometheus metrics.

- [Chapter 7, Monitoring your cluster using JMX](#)

## 14.1. CONSUMER LAG

Consumer lag indicates the difference in the rate of production and consumption of messages. Specifically, consumer lag for a given consumer group indicates the delay between the last message in the partition and the message being currently picked up by that consumer. The lag reflects the position of the consumer offset in relation to the end of the partition log.

This difference is sometimes referred to as the *delta* between the producer offset and consumer offset, the read and write positions in the Kafka broker topic partitions.

Suppose a topic streams 100 messages a second. A lag of 1000 messages between the producer offset (the topic partition head) and the last offset the consumer has read means a 10-second delay.

### The importance of monitoring consumer lag

For applications that rely on the processing of (near) real-time data, it is critical to monitor consumer lag to check that it does not become too big. The greater the lag becomes, the further the process moves from the real-time processing objective.

Consumer lag, for example, might be a result of consuming too much old data that has not been purged, or through unplanned shutdowns.

### Reducing consumer lag

Typical actions to reduce lag include:

- Scaling-up consumer groups by adding new consumers
- Increasing the retention time for a message to remain in a topic
- Adding more disk capacity to increase the message buffer

Actions to reduce consumer lag depend on the underlying infrastructure and the use cases AMQ Streams is supporting. For instance, a lagging consumer is less likely to benefit from the broker being able to service a fetch request from its disk cache. And in certain cases, it might be acceptable to automatically drop messages until a consumer has caught up.

## 14.2. KAFKA EXPORTER ALERTING RULE EXAMPLES

The sample alert notification rules specific to Kafka Exporter are as follows:

### UnderReplicatedPartition

An alert to warn that a topic is under-replicated and the broker is not replicating enough partitions. The default configuration is for an alert if there are one or more under-replicated partitions for a topic. The alert might signify that a Kafka instance is down or the Kafka cluster is overloaded. A planned restart of the Kafka broker may be required to restart the replication process.

### TooLargeConsumerGroupLag

An alert to warn that the lag on a consumer group is too large for a specific topic partition. The default configuration is 1000 records. A large lag might indicate that consumers are too slow and are falling behind the producers.

### NoMessageForTooLong

An alert to warn that a topic has not received messages for a period of time. The default configuration for the time period is 10 minutes. The delay might be a result of a configuration issue preventing a producer from publishing messages to the topic.

You can adapt alerting rules according to your specific needs.

### Additional resources

For more information about setting up alerting rules, see [Configuration](#) in the Prometheus documentation.

## 14.3. KAFKA EXPORTER METRICS

Lag information is exposed by Kafka Exporter as Prometheus metrics for presentation in Grafana.

Kafka Exporter exposes metrics data for brokers, topics, and consumer groups.

The data extracted is described here.

**Table 14.1. Broker metrics output**

Name	Information
<b>kafka_brokers</b>	Number of brokers in the Kafka cluster

**Table 14.2. Topic metrics output**

Name	Information
<b>kafka_topic_partitions</b>	Number of partitions for a topic
<b>kafka_topic_partition_current_offset</b>	Current topic partition offset for a broker
<b>kafka_topic_partition_oldest_offset</b>	Oldest topic partition offset for a broker
<b>kafka_topic_partition_in_sync_replica</b>	Number of in-sync replicas for a topic partition

Name	Information
<b>kafka_topic_partition_leader</b>	Leader broker ID of a topic partition
<b>kafka_topic_partition_leader_is_preferred</b>	Shows <b>1</b> if a topic partition is using the preferred broker
<b>kafka_topic_partition_replicas</b>	Number of replicas for this topic partition
<b>kafka_topic_partition_under_replicated_partition</b>	Shows <b>1</b> if a topic partition is under-replicated

Table 14.3. Consumer group metrics output

Name	Information
<b>kafka_consumergroup_current_offset</b>	Current topic partition offset for a consumer group
<b>kafka_consumergroup_lag</b>	Current approximate lag for a consumer group at a topic partition

## 14.4. RUNNING KAFKA EXPORTER

Kafka Exporter is provided with the download archive used for [Installing AMQ Streams](#).

You can run it to expose Prometheus metrics for presentation in a Grafana dashboard.

### Prerequisites

- [AMQ Streams is installed on the host](#)

This procedure assumes you already have access to a Grafana user interface and Prometheus is deployed and added as a data source.

### Procedure

1. Run the Kafka Exporter script using appropriate configuration parameter values.

```
./bin/kafka_exporter --kafka.server=<kafka-bootstrap-address>:9092 --kafka.version=2.4.0 --
<my-other-parameters>
```

The parameters require a double-hyphen convention, such as **--kafka.server**.

Option	Description	Default
<b>kafka.server</b>	Host/post address of the Kafka server.	<b>kafka:9092</b>

Option	Description	Default
<b>kafka.version</b>	Kafka broker version.	<b>1.0.0</b>
<b>group.filter</b>	A regular expression to specify the consumer groups to include in the metrics.	<b>.*</b> (all)
<b>topic.filter</b>	A regular expression to specify the topics to include in the metrics.	<b>.*</b> (all)
<b>sasl.&lt;parameter&gt;</b>	Parameters to enable and connect to the Kafka cluster using SASL/PLAIN authentication, with user name and password.	<b>false</b>
<b>tls.&lt;parameter&gt;</b>	Parameters to enable connect to the Kafka cluster using TLS authentication, with optional certificate and key.	<b>false</b>
<b>web.listen-address</b>	Port address to expose the metrics.	<b>:9308</b>
<b>web.telemetry-path</b>	Path for the exposed metrics.	<b>/metrics</b>
<b>log.level</b>	Logging configuration, to log messages with a given severity (debug, info, warn, error, fatal) or above.	<b>info</b>
<b>log.enable-sarama</b>	Boolean to enable Sarama logging, a Go client library used by the Kafka Exporter.	<b>false</b>

You can use **kafka\_exporter --help** for information on the properties.

2. Configure Prometheus to monitor the Kafka Exporter metrics.  
For more information on configuring Prometheus, see the [Prometheus documentation](#).
3. Enable Grafana to present the Kafka Exporter metrics data exposed by Prometheus.  
For more information, see [Presenting Kafka Exporter metrics in Grafana](#).

## 14.5. PRESENTING KAFKA EXPORTER METRICS IN GRAFANA

Using Kafka Exporter Prometheus metrics as a data source, you can create a dashboard of Grafana charts.

For example, from the metrics you can create the following Grafana charts:

- Message in per second (from topics)
- Message in per minute (from topics)
- Lag by consumer group
- Messages consumed per minute (by consumer groups)

When metrics data has been collected for some time, the Kafka Exporter charts are populated.

Use the Grafana charts to analyze lag and to check if actions to reduce lag are having an impact on an affected consumer group. If, for example, Kafka brokers are adjusted to reduce lag, the dashboard will show the *Lag by consumer group* chart going down and the *Messages consumed per minute* chart going up.

#### **Additional resources**

- [Example dashboard for Kafka Exporter](#)
- [Grafana documentation](#)

## CHAPTER 15. AMQ STREAMS AND KAFKA UPGRADES

AMQ Streams can be upgraded with no cluster downtime. Each version of AMQ Streams supports one or more versions of Apache Kafka: you can upgrade to a higher Kafka version as long as it is supported by your version of AMQ Streams. In some cases, you can also downgrade to a lower supported Kafka version.

Newer versions of AMQ Streams may support newer versions of Kafka, but you need to upgrade AMQ Streams *before* you can upgrade to a higher supported Kafka version.

### 15.1. UPGRADE PREREQUISITES

Before you begin the upgrade process, make sure that:

- AMQ Streams is installed. For instructions, see [Chapter 2, Getting started](#).
- You are familiar with any upgrade changes described in the [AMQ Streams 1.4 on Red Hat Enterprise Linux Release Notes](#).

### 15.2. UPGRADE PROCESS

Upgrading AMQ Streams is a two-stage process. To upgrade brokers and clients without downtime, you *must* complete the upgrade procedures in the following order:

1. Upgrade to the latest AMQ Streams version.
  - [Upgrading to AMQ Streams 1.4](#)
2. Upgrade all Kafka brokers and client applications to the latest Kafka version
  - [Upgrading Kafka](#)

### 15.3. KAFKA VERSIONS

Kafka's log message format version and inter-broker protocol version specify the log format version appended to messages and the version of protocol used in a cluster. As a result, the upgrade process involves making configuration changes to existing Kafka brokers and code changes to client applications (consumers and producers) to ensure the correct versions are used.

The following table shows the differences between Kafka versions:

Kafka version	Interbroker protocol version	Log message format version	ZooKeeper version
2.3.0	2.3	2.3	3.4.14
2.4.0	2.4	2.4	3.5.7

#### Message format version

When a producer sends a message to a Kafka broker, the message is encoded using a specific format. The format can change between Kafka releases, so messages include a version identifying which version of the format they were encoded with. You can configure a Kafka broker to convert messages from

newer format versions to a given older format version before the broker appends the message to the log.

In Kafka, there are two different methods for setting the message format version:

- The **message.format.version** property is set on topics.
- The **log.message.format.version** property is set on Kafka brokers.

The default value of **message.format.version** for a topic is defined by the **log.message.format.version** that is set on the Kafka broker. You can manually set the **message.format.version** of a topic by modifying its topic configuration.

The upgrade tasks in this section assume that the message format version is defined by the **log.message.format.version**.

## 15.4. UPGRADING TO AMQ STREAMS 1.4

The steps to upgrade your deployment to use AMQ Streams 1.4 are outlined in this section.

The availability of Kafka clusters managed by AMQ Streams is not affected by the upgrade operation.



### NOTE

Refer to the documentation supporting a specific version of AMQ Streams for information on how to upgrade to that version.

### 15.4.1. Upgrading ZooKeeper

This procedure describes how to upgrade ZooKeeper on a host machine.

**WARNING****Missing snapshots error when upgrading to ZooKeeper 3.5.7**

Kafka version 2.4 requires a new version of ZooKeeper, version 3.5.7. When upgrading to ZooKeeper 3.5.7, you might see the following error, which is related to a bug in previous ZooKeeper versions:

```
java.io.IOException: No snapshot found, but there are log entries. Something is broken!
```

This error occurs if the data directory for the ZooKeeper 3.5.7 cluster, **/var/lib/zookeeper/**, does not contain any snapshot files. This check was not performed in previous ZooKeeper versions.

If you see this error, you must set the **snapshot.trust.empty=true** configuration option in the ZooKeeper properties file, and then restart ZooKeeper. If ZooKeeper restarts successfully, you must then set the **snapshot.trust.empty=false** configuration option.

For more information, see the [Release Notes for AMQ Streams 1.4 on Red Hat Enterprise Linux](#) and [ZOOKEEPER-3056](#).

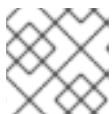
**Prerequisites**

- You are logged in to Red Hat Enterprise Linux as the **kafka** user.

**Procedure**

For each Kafka broker in your AMQ Streams cluster and one at a time:

- Download the AMQ Streams archive for AMQ Streams 1.4 from the [Customer Portal](#).

**NOTE**

If prompted, log in to your Red Hat account.

- On the command line, create a temporary directory and extract the contents of the **amq-streams-x.y.z-bin.zip** file:

```
mkdir /tmp/kafka
unzip amq-streams-x.y.z-bin.zip -d /tmp/kafka
```

- Delete the **libs**, **bin**, and **docs** directories from your existing installation:

```
rm -rf /opt/kafka/libs /opt/kafka/bin /opt/kafka/docs
```

- Copy the **libs**, **bin**, and **docs** directories from the temporary directory:



```
cp -r /tmp/kafka/kafka_y.y-x.x.x/libs /opt/kafka/
cp -r /tmp/kafka/kafka_y.y-x.x.x/bin /opt/kafka/
cp -r /tmp/kafka/kafka_y.y-x.x.x/docs /opt/kafka/
```

5. Delete the temporary directory:

```
rm -rf /tmp/kafka
```

6. Add the following configuration options to the ZooKeeper configuration file, **/opt/kafka/config/zookeeper.properties**:

```
reconfigEnabled=true 1
standaloneEnabled=false 2
```

- 1** Enables dynamic reconfiguration. This is required to add or remove servers to a ZooKeeper 3.5.7 cluster.
- 2** Disables standalone mode, where ZooKeeper runs with only one server. For production use cases, always run ZooKeeper with more than one server.

These configuration options were added in ZooKeeper 3.5.7.

7. In the ZooKeeper configuration file, reformat existing server addresses to match the new address format introduced in ZooKeeper 3.5.0. For example:

```
server.1=172.17.0.1:2888:3888:participant;172.17.0.1:2181
server.2=172.17.0.2:2888:3888:participant;172.17.0.2:2181
server.3=172.17.0.3:2888:3888:participant;172.17.0.3:2181
```



#### NOTE

The client port is now specified in the server address instead of the static ZooKeeper configuration file.

8. Restart ZooKeeper:

```
/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties
```

## 15.4.2. Upgrading Kafka brokers

This procedure describes how to upgrade Kafka brokers on a host machine.

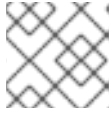
### Prerequisites

- You are logged in to Red Hat Enterprise Linux as the **kafka** user.

### Procedure

For each Kafka broker in your AMQ Streams cluster and one at a time:

1. Download the AMQ Streams archive from the [Customer Portal](#).

**NOTE**

If prompted, log in to your Red Hat account.

2. On the command line, create a temporary directory and extract the contents of the **amq-streams-x.y.z-bin.zip** file.

```
mkdir /tmp/kafka
unzip amq-streams-x.y.z-bin.zip -d /tmp/kafka
```

3. Delete the **libs**, **bin**, and **docs** directories from your existing installation:

```
rm -rf /opt/kafka/libs /opt/kafka/bin /opt/kafka/docs
```

4. Copy the **libs**, **bin**, and **docs** directories from the temporary directory:

```
cp -r /tmp/kafka/kafka_y.y-x.x.x/libs /opt/kafka/
cp -r /tmp/kafka/kafka_y.y-x.x.x/bin /opt/kafka/
cp -r /tmp/kafka/kafka_y.y-x.x.x/docs /opt/kafka/
```

5. Delete the temporary directory.

```
rm -r /tmp/kafka
```

6. In a text editor, open the broker properties file, commonly stored in the **/opt/kafka/config/** directory.

7. Temporarily override the default inter-broker protocol and message format versions for Kafka 2.4.0 by adding or updating the following properties in the file:

```
inter.broker.protocol.version=2.3
log.message.format.version=2.3
```

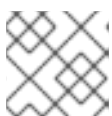
This configures the Kafka broker to process data using the previous inter-broker protocol (**2.3**) and message format versions.

8. On the command line, stop the Kafka broker that you modified:

```
/opt/kafka/bin/kafka-server-stop.sh
jcmd | grep kafka
```

9. Restart the Kafka broker that you modified:

```
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
```

**NOTE**

The Kafka broker will start using the binaries for the latest Kafka version.

10. Verify that the restarted Kafka broker has caught up with the partition replicas it is following. Use the **kafka-topics.sh** tool to ensure that all replicas contained in the broker are back in sync. For instructions, see [Listing and describing topics](#).

### 15.4.3. Upgrading Kafka Connect

This procedure describes how to upgrade a Kafka Connect cluster on a host machine.

Kafka Connect is a client application and should be included in your chosen strategy for upgrading clients. For more information, see [Strategies for upgrading clients](#).

#### Prerequisites

- You are logged in to Red Hat Enterprise Linux as the **kafka** user.

#### Procedure

For each Kafka broker in your AMQ Streams cluster and one at a time:

- Download the AMQ Streams archive from the [Customer Portal](#).



#### NOTE

If prompted, log in to your Red Hat account.

- On the command line, create a temporary directory and extract the contents of the **amq-streams-x.y.z-bin.zip** file.

```
mkdir /tmp/kafka
unzip amq-streams-x.y.z-bin.zip -d /tmp/kafka
```

- Delete the **libs**, **bin**, and **docs** directories from your existing installation:

```
rm -rf /opt/kafka/libs /opt/kafka/bin /opt/kafka/docs
```

- Copy the **libs**, **bin**, and **docs** directories from the temporary directory:

```
cp -r /tmp/kafka/kafka_y.y-x.x.x/libs /opt/kafka/
cp -r /tmp/kafka/kafka_y.y-x.x.x/bin /opt/kafka/
cp -r /tmp/kafka/kafka_y.y-x.x.x/docs /opt/kafka/
```

- Delete the temporary directory.

```
rm -r /tmp/kafka
```

- Start Kafka Connect in either standalone or distributed mode.

- To start in standalone mode, run the **connect-standalone.sh** script. Specify the Kafka Connect standalone configuration file and the configuration files of your Kafka Connect connectors.

```
su - kafka
/opt/kafka/bin/connect-standalone.sh /opt/kafka/config/connect-standalone.properties
connector1.properties
[connector2.properties ...]
```

- To start in distributed mode, start the Kafka Connect workers with the **/opt/kafka/config/connect-distributed.properties** configuration file on all Kafka Connect nodes:

```
su - kafka  
/opt/kafka/bin/connect-distributed.sh /opt/kafka/config/connect-distributed.properties
```

7. Verify that Kafka Connect is running:

- In standalone mode:

```
jcmd | grep ConnectStandalone
```

- In distributed mode:

```
jcmd | grep ConnectDistributed
```

8. Verify that Kafka Connect is producing and consuming data as expected.

### Additional resources

- [Running Kafka Connect in standalone mode](#)
- [Running distributed Kafka Connect](#)
- [Strategies for upgrading clients](#)

## 15.5. UPGRADING KAFKA

After you have upgraded your binaries to use the latest version of AMQ Streams, you can upgrade your brokers and clients to use a higher supported version of Kafka.

Take care to follow the steps in the correct order:

1. [Section 15.5.1, "Upgrading Kafka brokers to use the new inter-broker protocol version"](#)
2. [Section 15.5.3, "Upgrading client applications to the new Kafka version"](#)
3. [Section 15.5.5, "Upgrading Kafka brokers to use the new message format version"](#)

### Additional resources

- [Section 15.4, "Upgrading to AMQ Streams 1.4"](#)

### 15.5.1. Upgrading Kafka brokers to use the new inter-broker protocol version

Manually configure and restart all Kafka brokers to use the new inter-broker protocol version. After performing these steps, data is transmitted between the Kafka brokers using the new inter-broker protocol version.

Messages received are still appended to the message logs in the earlier message format version.

**WARNING**

Downgrading AMQ Streams is not possible after completing this procedure.

**Prerequisites**

- You have updated the ZooKeeper binaries.
- You have upgraded all Kafka brokers to AMQ Streams 1.4
- You are logged in to Red Hat Enterprise Linux as the **kafka** user.

**Procedure**

For each Kafka broker in your AMQ Streams cluster and one at a time:

1. In a text editor, open the broker properties file for the Kafka broker you want to update. Broker properties files are commonly stored in the **/opt/kafka/config/** directory.
2. Set the **inter.broker.protocol.version** to **2.4**.

```
inter.broker.protocol.version=2.4
```

3. On the command line, stop the Kafka broker that you modified:

```
/opt/kafka/bin/kafka-server-stop.sh
jcmd | grep kafka
```

4. Restart the Kafka broker that you modified:

```
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
```

5. Verify that the restarted Kafka broker has caught up with the partition replicas it is following. Use the **kafka-topics.sh** tool to ensure that all replicas contained in the broker are back in sync. For instructions, see [Listing and describing topics](#).

**15.5.2. Strategies for upgrading clients**

The best approach to upgrading your client applications (including Kafka Connect connectors) depends on your particular circumstances.

Consuming applications need to receive messages in a message format that they understand. You can ensure that this is the case in one of two ways:

- By upgrading all the consumers for a topic *before* upgrading any of the producers.
- By having the brokers down-convert messages to an older format.

Using broker down-conversion puts extra load on the brokers, so it is not ideal to rely on down-conversion for all topics for a prolonged period of time. For brokers to perform optimally they should not be down converting messages at all.

Broker down-conversion is configured in two ways:

- The topic-level **message.format.version** configures it for a single topic.
- The broker-level **log.message.format.version** is the default for topics that do not have the topic-level **message.format.version** configured.

Messages published to a topic in a new-version format will be visible to consumers, because brokers perform down-conversion when they receive messages from producers, not when they are sent to consumers.

There are a number of strategies you can use to upgrade your clients:

### Consumers first

1. Upgrade all the consuming applications.
2. Change the broker-level **log.message.format.version** to the new version.
3. Upgrade all the producing applications.  
This strategy is straightforward, and avoids any broker down-conversion. However, it assumes that all consumers in your organization can be upgraded in a coordinated way, and it does not work for applications that are both consumers and producers. There is also a risk that, if there is a problem with the upgraded clients, new-format messages might get added to the message log so that you cannot revert to the previous consumer version.

### Per-topic consumers first

For each topic:

1. Upgrade all the consuming applications.
2. Change the topic-level **message.format.version** to the new version.
3. Upgrade all the producing applications.  
This strategy avoids any broker down-conversion, and means you can proceed on a topic-by-topic basis. It does not work for applications that are both consumers and producers of the same topic. Again, it has the risk that, if there is a problem with the upgraded clients, new-format messages might get added to the message log.

### Per-topic consumers first, with down conversion

For each topic:

1. Change the topic-level **message.format.version** to the old version (or rely on the topic defaulting to the broker-level **log.message.format.version**).
2. Upgrade all the consuming and producing applications.
3. Verify that the upgraded applications function correctly.
4. Change the topic-level **message.format.version** to the new version.  
This strategy requires broker down-conversion, but the load on the brokers is minimized because it is only required for a single topic (or small group of topics) at a time. It also works for applications that are both consumers and producers of the same topic. This approach ensures that the upgraded producers and consumers are working correctly before you commit to using the new message format version.

The main drawback of this approach is that it can be complicated to manage in a cluster with many topics and applications.

Other strategies for upgrading client applications are also possible.



#### NOTE

It is also possible to apply multiple strategies. For example, for the first few applications and topics the "per-topic consumers first, with down conversion" strategy can be used. When this has proved successful another, more efficient strategy can be considered acceptable to use instead.

### 15.5.3. Upgrading client applications to the new Kafka version

This procedure describes one possible approach to upgrading your client applications to the Kafka version used for AMQ Streams 1.4.

The procedure is based on the "per-topic consumers first, with down conversion" approach outlined in [Strategies for upgrading clients](#).

Client applications include producers, consumers, Kafka Connect, Kafka Streams applications, and MirrorMaker.

#### Prerequisites

- You have updated the ZooKeeper binaries.
- You have upgraded all Kafka brokers to AMQ Streams 1.4.
- You have configured Kafka brokers to use the new inter-broker protocol version.
- You are logged in to Red Hat Enterprise Linux as the **kafka** user.

#### Procedure

For each topic:

1. On the command line, set the **message.format.version** configuration option to **2.3**.

```
bin/kafka-configs.sh --zookeeper <ZooKeeperAddress> --entity-type topics --entity-name
<TopicName> --alter --add-config message.format.version=2.3
```

2. Upgrade all the consumers and producers for the topic.
3. Optionally, to upgrade consumers and Kafka Streams applications to use the *incremental cooperative rebalance* protocol, which was added in Kafka 2.4.0, see [Section 15.5.4, "Upgrading consumers and Kafka Streams applications to cooperative rebalancing"](#).
4. Verify that the upgraded applications function correctly.
5. Change the topic's **message.format.version** configuration option to **2.4**.

```
bin/kafka-configs.sh --zookeeper <ZooKeeperAddress> --entity-type topics --entity-name
<TopicName> --alter --add-config message.format.version=2.4
```

## Additional resources

- [Strategies for upgrading clients](#)

### 15.5.4. Upgrading consumers and Kafka Streams applications to cooperative rebalancing

You can upgrade Kafka consumers and Kafka Streams applications to use the *incremental cooperative rebalance* protocol for partition rebalances instead of the default *eager rebalance* protocol. The new protocol was added in Kafka 2.4.0.

Consumers keep their partition assignments in a cooperative rebalance and only revoke them at the end of the process, if needed to achieve a balanced cluster. This reduces the unavailability of the consumer group or Kafka Streams application.



#### NOTE

Upgrading to the incremental cooperative rebalance protocol is optional. The eager rebalance protocol is still supported.

#### Prerequisites

- [Section 15.4, “Upgrading to AMQ Streams 1.4”](#)
- [Section 15.5.1, “Upgrading Kafka brokers to use the new inter-broker protocol version”](#)
- [Section 15.5.3, “Upgrading client applications to the new Kafka version”](#)

#### Procedure

**To upgrade a Kafka consumer to use the incremental cooperative rebalance protocol:**

1. Replace the Kafka clients **.jar** file with the new version.
2. In the consumer configuration, append **cooperative-sticky** to the **partition.assignment.strategy**. For example, if the **range** strategy is set, change the configuration to **range, cooperative-sticky**.
3. Restart each consumer in the group in turn, waiting for the consumer to rejoin the group after each restart.
4. Reconfigure each consumer in the group by removing the earlier **partition.assignment.strategy** from the consumer configuration, leaving only the **cooperative-sticky** strategy.
5. Restart each consumer in the group in turn, waiting for the consumer to rejoin the group after each restart.

**To upgrade a Kafka Streams application to use the incremental cooperative rebalance protocol:**

1. Replace the Kafka Streams **.jar** file with the new version.
2. In the Kafka Streams configuration, set the **upgrade.from** configuration parameter to the Kafka version you are upgrading from (for example, 2.3).
3. Restart each of the stream processors (nodes) in turn.



4. Remove the **upgrade.from** configuration parameter from the Kafka Streams configuration.
5. Restart each consumer in the group in turn.

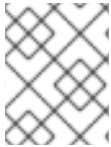
### Additional resources

- [Notable changes in 2.4.0](#) in the Apache Kafka documentation.

### 15.5.5. Upgrading Kafka brokers to use the new message format version

When client applications have been upgraded, you can update the Kafka brokers to use the new message format version.

If you did *not* modify topic configurations when you upgraded your client applications to use the Kafka version required for AMQ Streams 1.4, the Kafka brokers are now converting messages down to the previous message format version, which can cause a reduction in performance. Therefore, it is important that you update all Kafka brokers to use the new message format version as soon as possible.



#### NOTE

Update and restart the Kafka brokers one-by-one. Before you restart a modified broker, stop the broker you configured and restarted previously.

### Prerequisites

- You have updated the ZooKeeper binaries.
- You have upgraded all Kafka brokers to AMQ Streams 1.4.
- You have configured Kafka brokers to use the new inter-broker protocol version.
- You have upgraded supported client applications that consume messages from topics for which the **message.format.version** property is *not* explicitly configured at the topic level.
- You are logged in to Red Hat Enterprise Linux as the **kafka** user.

### Procedure

For each Kafka broker in your AMQ Streams cluster and one at a time:

1. In a text editor, open the broker properties file for the Kafka broker you want to update. Broker properties files are commonly stored in the **/opt/kafka/config/** directory.
2. Set the **log.message.format.version** to **2.4**.
 

```
log.message.format.version=2.4
```
3. On the command line, stop the Kafka broker that you most recently modified and restarted as part of this procedure. If you are modifying the first Kafka broker in this procedure, go to step four.

```
/opt/kafka/bin/kafka-server-stop.sh
jcmd | grep kafka
```

4. Restart the Kafka broker whose configuration you modified in step two:

```
| /opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/config/server.properties
```

5. Verify that the restarted Kafka broker has caught up with the partition replicas it is following. Use the **kafka-topics.sh** tool to ensure that all replicas contained in the broker are back in sync. For instructions, see [Listing and describing topics](#).

/home/pmellor/Documentation/amq-streams/books-ocp/modules/using/ref-kafka-versions.adoc

## APPENDIX A. BROKER CONFIGURATION PARAMETERS

### zookeeper.connect

**Type:** string

**Importance:** high

**Dynamic update:** read-only

Specifies the ZooKeeper connection string in the form **hostname:port** where host and port are the host and port of a ZooKeeper server. To allow connecting through other ZooKeeper nodes when that ZooKeeper machine is down you can also specify multiple hosts in the form

**hostname1:port1,hostname2:port2,hostname3:port3**. The server can also have a ZooKeeper chroot path as part of its ZooKeeper connection string which puts its data under some path in the global ZooKeeper namespace. For example to give a chroot path of **/chroot/path** you would give the connection string as **hostname1:port1,hostname2:port2,hostname3:port3/chroot/path**.

### advertised.host.name

**Type:** string

**Default:** null

**Importance:** high

**Dynamic update:** read-only

DEPRECATED: only used when **advertised.listeners** or **listeners** are not set. Use **advertised.listeners** instead. Hostname to publish to ZooKeeper for clients to use. In IaaS environments, this may need to be different from the interface to which the broker binds. If this is not set, it will use the value for **host.name** if configured. Otherwise it will use the value returned from `java.net.InetAddress.getCanonicalHostName()`.

### advertised.listeners

**Type:** string

**Default:** null

**Importance:** high

**Dynamic update:** per-broker

Listeners to publish to ZooKeeper for clients to use, if different than the **listeners** config property. In IaaS environments, this may need to be different from the interface to which the broker binds. If this is not set, the value for **listeners** will be used. Unlike **listeners** it is not valid to advertise the 0.0.0.0 meta-address.

### advertised.port

**Type:** int

**Default:** null

**Importance:** high

**Dynamic update:** read-only

DEPRECATED: only used when **advertised.listeners** or **listeners** are not set. Use **advertised.listeners** instead. The port to publish to ZooKeeper for clients to use. In IaaS environments, this may need to be different from the port to which the broker binds. If this is not set, it will publish the same port that the broker binds to.

### auto.create.topics.enable

**Type:** boolean

**Default:** true

**Importance:** high

**Dynamic update:** read-only

Enable auto creation of topic on the server.

**auto.leader.rebalance.enable****Type:** boolean**Default:** true**Importance:** high**Dynamic update:** read-only

Enables auto leader balancing. A background thread checks and triggers leader balance if required at regular intervals.

**background.threads****Type:** int**Default:** 10**Valid Values:** [1,...]**Importance:** high**Dynamic update:** cluster-wide

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

**broker.id****Type:** int**Default:** -1**Importance:** high**Dynamic update:** read-only

The broker id for this server. If unset, a unique broker id will be generated. To avoid conflicts between zookeeper generated broker id's and user configured broker id's, generated broker ids start from reserved.broker.max.id + 1.

**compression.type****Type:** string**Default:** producer**Importance:** high**Dynamic update:** cluster-wide

Specify the final compression type for a given topic. This configuration accepts the standard compression codecs ('gzip', 'snappy', 'lz4', 'zstd'). It additionally accepts 'uncompressed' which is equivalent to no compression; and 'producer' which means retain the original compression codec set by the producer.

**control.plane.listener.name****Type:** string**Default:** null**Importance:** high**Dynamic update:** read-only

Name of listener used for communication between controller and brokers. Broker will use the control.plane.listener.name to locate the endpoint in listeners list, to listen for connections from the controller. For example, if a broker's config is : listeners = INTERNAL://192.1.1.8:9092, EXTERNAL://10.1.1.5:9093, CONTROLLER://192.1.1.8:9094 listener.security.protocol.map = INTERNAL:PLAINTEXT, EXTERNAL:SSL, CONTROLLER:SSL control.plane.listener.name = CONTROLLER On startup, the broker will start listening on "192.1.1.8:9094" with security protocol "SSL". On controller side, when it discovers a broker's published endpoints through zookeeper, it will use the control.plane.listener.name to find the endpoint, which it will use to establish connection to the broker. For example, if the broker's published endpoints on zookeeper are : "endpoints" : ["INTERNAL://broker1.example.com:9092", "EXTERNAL://broker1.example.com:9093", "CONTROLLER and the controller's config is : listener.security.protocol.map = INTERNAL:PLAINTEXT, EXTERNAL:SSL, CONTROLLER:SSL control.plane.listener.name = CONTROLLER then controller

will use "broker1.example.com:9094" with security protocol "SSL" to connect to the broker. If not explicitly configured, the default value will be null and there will be no dedicated endpoints for controller connections.

### **delete.topic.enable**

**Type:** boolean

**Default:** true

**Importance:** high

**Dynamic update:** read-only

Enables delete topic. Delete topic through the admin tool will have no effect if this config is turned off.

### **host.name**

**Type:** string

**Default:** ""

**Importance:** high

**Dynamic update:** read-only

DEPRECATED: only used when **listeners** is not set. Use **listeners** instead. hostname of broker. If this is set, it will only bind to this address. If this is not set, it will bind to all interfaces.

### **leader.imbalance.check.interval.seconds**

**Type:** long

**Default:** 300

**Importance:** high

**Dynamic update:** read-only

The frequency with which the partition rebalance check is triggered by the controller.

### **leader.imbalance.per.broker.percentage**

**Type:** int

**Default:** 10

**Importance:** high

**Dynamic update:** read-only

The ratio of leader imbalance allowed per broker. The controller would trigger a leader balance if it goes above this value per broker. The value is specified in percentage.

### **listeners**

**Type:** string

**Default:** null

**Importance:** high

**Dynamic update:** per-broker

Listener List - Comma-separated list of URIs we will listen on and the listener names. If the listener name is not a security protocol, listener.security.protocol.map must also be set. Specify hostname as 0.0.0.0 to bind to all interfaces. Leave hostname empty to bind to default interface. Examples of legal listener lists: PLAINTEXT://myhost:9092,SSL://:9091  
CLIENT://0.0.0.0:9092,REPLICATION://localhost:9093.

### **log.dir**

**Type:** string

**Default:** /tmp/kafka-logs

**Importance:** high

**Dynamic update:** read-only

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

**log.dirs**

**Type:** string

**Default:** null

**Importance:** high

**Dynamic update:** read-only

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

**log.flush.interval.messages**

**Type:** long

**Default:** 9223372036854775807

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** cluster-wide

The number of messages accumulated on a log partition before messages are flushed to disk.

**log.flush.interval.ms**

**Type:** long

**Default:** null

**Importance:** high

**Dynamic update:** cluster-wide

The maximum time in ms that a message in any topic is kept in memory before flushed to disk. If not set, the value in log.flush.scheduler.interval.ms is used.

**log.flush.offset.checkpoint.interval.ms**

**Type:** int

**Default:** 60000

**Valid Values:** [0,...]

**Importance:** high

**Dynamic update:** read-only

The frequency with which we update the persistent record of the last flush which acts as the log recovery point.

**log.flush.scheduler.interval.ms**

**Type:** long

**Default:** 9223372036854775807

**Importance:** high

**Dynamic update:** read-only

The frequency in ms that the log flusher checks whether any log needs to be flushed to disk.

**log.flush.start.offset.checkpoint.interval.ms**

**Type:** int

**Default:** 60000

**Valid Values:** [0,...]

**Importance:** high

**Dynamic update:** read-only

The frequency with which we update the persistent record of log start offset.

**log.retention.bytes**

**Type:** long

**Default:** -1

**Importance:** high

**Dynamic update:** cluster-wide

The maximum size of the log before deleting it.

### **log.retention.hours**

**Type:** int

**Default:** 168

**Importance:** high

**Dynamic update:** read-only

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

### **log.retention.minutes**

**Type:** int

**Default:** null

**Importance:** high

**Dynamic update:** read-only

The number of minutes to keep a log file before deleting it (in minutes), secondary to log.retention.ms property. If not set, the value in log.retention.hours is used.

### **log.retention.ms**

**Type:** long

**Default:** null

**Importance:** high

**Dynamic update:** cluster-wide

The number of milliseconds to keep a log file before deleting it (in milliseconds), If not set, the value in log.retention.minutes is used. If set to -1, no time limit is applied.

### **log.roll.hours**

**Type:** int

**Default:** 168

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

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

### **log.roll.jitter.hours**

**Type:** int

**Default:** 0

**Valid Values:** [0,...]

**Importance:** high

**Dynamic update:** read-only

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

### **log.roll.jitter.ms**

**Type:** long

**Default:** null

**Importance:** high

**Dynamic update:** cluster-wide

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

**log.roll.ms****Type:** long**Default:** null**Importance:** high**Dynamic update:** cluster-wide

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

**log.segment.bytes****Type:** int**Default:** 1073741824**Valid Values:** [14,...]**Importance:** high**Dynamic update:** cluster-wide

The maximum size of a single log file.

**log.segment.delete.delay.ms****Type:** long**Default:** 60000**Valid Values:** [0,...]**Importance:** high**Dynamic update:** cluster-wide

The amount of time to wait before deleting a file from the filesystem.

**message.max.bytes****Type:** int**Default:** 1000012**Valid Values:** [0,...]**Importance:** high**Dynamic update:** cluster-wide

The largest record batch size allowed by Kafka. If this is increased and there are consumers older than 0.10.2, the consumers' fetch size must also be increased so that they can fetch record batches this large.

In the latest message format version, records are always grouped into batches for efficiency. In previous message format versions, uncompressed records are not grouped into batches and this limit only applies to a single record in that case.

This can be set per topic with the topic level **max.message.bytes** config.

**min.insync.replicas****Type:** int**Default:** 1**Valid Values:** [1,...]**Importance:** high**Dynamic update:** cluster-wide

When a producer sets `acks` to "all" (or "-1"), `min.insync.replicas` specifies the minimum number of replicas that must acknowledge a write for the write to be considered successful. If this minimum cannot be met, then the producer will raise an exception (either `NotEnoughReplicas` or `NotEnoughReplicasAfterAppend`). When used together, `min.insync.replicas` and `acks` allow you to enforce greater durability guarantees. A typical scenario would be to create a topic with a replication factor of 3, set `min.insync.replicas` to 2, and produce with `acks` of "all". This will ensure that the producer raises an exception if a majority of replicas do not receive a write.



**num.io.threads**

**Type:** int

**Default:** 8

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** cluster-wide

The number of threads that the server uses for processing requests, which may include disk I/O.

**num.network.threads**

**Type:** int

**Default:** 3

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** cluster-wide

The number of threads that the server uses for receiving requests from the network and sending responses to the network.

**num.recovery.threads.per.data.dir**

**Type:** int

**Default:** 1

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** cluster-wide

The number of threads per data directory to be used for log recovery at startup and flushing at shutdown.

**num.replica.alter.log.dirs.threads**

**Type:** int

**Default:** null

**Importance:** high

**Dynamic update:** read-only

The number of threads that can move replicas between log directories, which may include disk I/O.

**num.replica.fetchers**

**Type:** int

**Default:** 1

**Importance:** high

**Dynamic update:** cluster-wide

Number of fetcher threads used to replicate messages from a source broker. Increasing this value can increase the degree of I/O parallelism in the follower broker.

**offset.metadata.max.bytes**

**Type:** int

**Default:** 4096

**Importance:** high

**Dynamic update:** read-only

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

**offsets.commit.required.acks**

**Type:** short

**Default:** -1

**Importance:** high

**Dynamic update:** read-only

The required acks before the commit can be accepted. In general, the default (-1) should not be overridden.

#### **offsets.commit.timeout.ms**

**Type:** int

**Default:** 5000

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

Offset commit will be delayed until all replicas for the offsets topic receive the commit or this timeout is reached. This is similar to the producer request timeout.

#### **offsets.load.buffer.size**

**Type:** int

**Default:** 5242880

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

Batch size for reading from the offsets segments when loading offsets into the cache (soft-limit, overridden if records are too large).

#### **offsets.retention.check.interval.ms**

**Type:** long

**Default:** 600000

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

Frequency at which to check for stale offsets.

#### **offsets.retention.minutes**

**Type:** int

**Default:** 10080

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

After a consumer group loses all its consumers (i.e. becomes empty) its offsets will be kept for this retention period before getting discarded. For standalone consumers (using manual assignment), offsets will be expired after the time of last commit plus this retention period.

#### **offsets.topic.compression.codec**

**Type:** int

**Default:** 0

**Importance:** high

**Dynamic update:** read-only

Compression codec for the offsets topic - compression may be used to achieve "atomic" commits.

#### **offsets.topic.num.partitions**

**Type:** int

**Default:** 50

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

The number of partitions for the offset commit topic (should not change after deployment).

### **offsets.topic.replication.factor**

**Type:** short

**Default:** 3

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

The replication factor for the offsets topic (set higher to ensure availability). Internal topic creation will fail until the cluster size meets this replication factor requirement.

### **offsets.topic.segment.bytes**

**Type:** int

**Default:** 104857600

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

The offsets topic segment bytes should be kept relatively small in order to facilitate faster log compaction and cache loads.

### **port**

**Type:** int

**Default:** 9092

**Importance:** high

**Dynamic update:** read-only

DEPRECATED: only used when **listeners** is not set. Use **listeners** instead. the port to listen and accept connections on.

### **queued.max.requests**

**Type:** int

**Default:** 500

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

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

### **quota.consumer.default**

**Type:** long

**Default:** 9223372036854775807

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

DEPRECATED: Used only when dynamic default quotas are not configured for <user, <client-id> or <user, client-id> in Zookeeper. Any consumer distinguished by clientId/consumer group will get throttled if it fetches more bytes than this value per-second.

### **quota.producer.default**

**Type:** long

**Default:** 9223372036854775807

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

DEPRECATED: Used only when dynamic default quotas are not configured for <user>, <client-id> or <user, client-id> in Zookeeper. Any producer distinguished by clientId will get throttled if it produces more bytes than this value per-second.

### **replica.fetch.min.bytes**

**Type:** int

**Default:** 1

**Importance:** high

**Dynamic update:** read-only

Minimum bytes expected for each fetch response. If not enough bytes, wait up to replicaMaxWaitTimeMs.

### **replica.fetch.wait.max.ms**

**Type:** int

**Default:** 500

**Importance:** high

**Dynamic update:** read-only

max wait time for each fetcher request issued by follower replicas. This value should always be less than the replica.lag.time.max.ms at all times to prevent frequent shrinking of ISR for low throughput topics.

### **replica.high.watermark.checkpoint.interval.ms**

**Type:** long

**Default:** 5000

**Importance:** high

**Dynamic update:** read-only

The frequency with which the high watermark is saved out to disk.

### **replica.lag.time.max.ms**

**Type:** long

**Default:** 10000

**Importance:** high

**Dynamic update:** read-only

If a follower hasn't sent any fetch requests or hasn't consumed up to the leaders log end offset for at least this time, the leader will remove the follower from isr.

### **replica.socket.receive.buffer.bytes**

**Type:** int

**Default:** 65536

**Importance:** high

**Dynamic update:** read-only

The socket receive buffer for network requests.

### **replica.socket.timeout.ms**

**Type:** int

**Default:** 30000

**Importance:** high

**Dynamic update:** read-only

The socket timeout for network requests. Its value should be at least replica.fetch.wait.max.ms.

**request.timeout.ms**

**Type:** int

**Default:** 30000

**Importance:** high

**Dynamic update:** read-only

The configuration controls the maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.

**socket.receive.buffer.bytes**

**Type:** int

**Default:** 102400

**Importance:** high

**Dynamic update:** read-only

The SO\_RCVBUF buffer of the socket server sockets. If the value is -1, the OS default will be used.

**socket.request.max.bytes**

**Type:** int

**Default:** 104857600

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

The maximum number of bytes in a socket request.

**socket.send.buffer.bytes**

**Type:** int

**Default:** 102400

**Importance:** high

**Dynamic update:** read-only

The SO\_SNDBUF buffer of the socket server sockets. If the value is -1, the OS default will be used.

**transaction.max.timeout.ms**

**Type:** int

**Default:** 900000

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

The maximum allowed timeout for transactions. If a client's requested transaction time exceed this, then the broker will return an error in `InitProducerIdRequest`. This prevents a client from too large of a timeout, which can stall consumers reading from topics included in the transaction.

**transaction.state.log.load.buffer.size**

**Type:** int

**Default:** 5242880

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

Batch size for reading from the transaction log segments when loading producer ids and transactions into the cache (soft-limit, overridden if records are too large).

**transaction.state.log.min.isr**

**Type:** int  
**Default:** 2  
**Valid Values:** [1,...]  
**Importance:** high  
**Dynamic update:** read-only  
Overridden min.insync.replicas config for the transaction topic.

#### **transaction.state.log.num.partitions**

**Type:** int  
**Default:** 50  
**Valid Values:** [1,...]  
**Importance:** high  
**Dynamic update:** read-only  
The number of partitions for the transaction topic (should not change after deployment).

#### **transaction.state.log.replication.factor**

**Type:** short  
**Default:** 3  
**Valid Values:** [1,...]  
**Importance:** high  
**Dynamic update:** read-only  
The replication factor for the transaction topic (set higher to ensure availability). Internal topic creation will fail until the cluster size meets this replication factor requirement.

#### **transaction.state.log.segment.bytes**

**Type:** int  
**Default:** 104857600  
**Valid Values:** [1,...]  
**Importance:** high  
**Dynamic update:** read-only  
The transaction topic segment bytes should be kept relatively small in order to facilitate faster log compaction and cache loads.

#### **transactional.id.expiration.ms**

**Type:** int  
**Default:** 604800000  
**Valid Values:** [1,...]  
**Importance:** high  
**Dynamic update:** read-only  
The maximum amount of time in ms that the transaction coordinator will wait before proactively expire a producer's transactional id without receiving any transaction status updates from it.

#### **unclean.leader.election.enable**

**Type:** boolean  
**Default:** false  
**Importance:** high  
**Dynamic update:** cluster-wide  
Indicates whether to enable replicas not in the ISR set to be elected as leader as a last resort, even though doing so may result in data loss.

#### **zookeeper.connection.timeout.ms**

**Type:** int

**Default:** null

**Importance:** high

**Dynamic update:** read-only

The max time that the client waits to establish a connection to zookeeper. If not set, the value in `zookeeper.session.timeout.ms` is used.

### **zookeeper.max.in.flight.requests**

**Type:** int

**Default:** 10

**Valid Values:** [1,...]

**Importance:** high

**Dynamic update:** read-only

The maximum number of unacknowledged requests the client will send to Zookeeper before blocking.

### **zookeeper.session.timeout.ms**

**Type:** int

**Default:** 6000

**Importance:** high

**Dynamic update:** read-only

Zookeeper session timeout.

### **zookeeper.set.acl**

**Type:** boolean

**Default:** false

**Importance:** high

**Dynamic update:** read-only

Set client to use secure ACLs.

### **broker.id.generation.enable**

**Type:** boolean

**Default:** true

**Importance:** medium

**Dynamic update:** read-only

Enable automatic broker id generation on the server. When enabled the value configured for `reserved.broker.max.id` should be reviewed.

### **broker.rack**

**Type:** string

**Default:** null

**Importance:** medium

**Dynamic update:** read-only

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

Examples: **RACK1**, **us-east-1d**.

### **connections.max.idle.ms**

**Type:** long

**Default:** 600000

**Importance:** medium

**Dynamic update:** read-only

Idle connections timeout: the server socket processor threads close the connections that idle more than this.

**connections.max.reauth.ms****Type:** long**Default:** 0**Importance:** medium**Dynamic update:** read-only

When explicitly set to a positive number (the default is 0, not a positive number), a session lifetime that will not exceed the configured value will be communicated to v2.2.0 or later clients when they authenticate. The broker will disconnect any such connection that is not re-authenticated within the session lifetime and that is then subsequently used for any purpose other than re-authentication. Configuration names can optionally be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.oauthbearer.connections.max.reauth.ms=3600000`.

**controlled.shutdown.enable****Type:** boolean**Default:** true**Importance:** medium**Dynamic update:** read-only

Enable controlled shutdown of the server.

**controlled.shutdown.max.retries****Type:** int**Default:** 3**Importance:** medium**Dynamic update:** read-only

Controlled shutdown can fail for multiple reasons. This determines the number of retries when such failure happens.

**controlled.shutdown.retry.backoff.ms****Type:** long**Default:** 5000**Importance:** medium**Dynamic update:** read-only

Before each retry, the system needs time to recover from the state that caused the previous failure (Controller fail over, replica lag etc). This config determines the amount of time to wait before retrying.

**controller.socket.timeout.ms****Type:** int**Default:** 30000**Importance:** medium**Dynamic update:** read-only

The socket timeout for controller-to-broker channels.

**default.replication.factor****Type:** int**Default:** 1**Importance:** medium**Dynamic update:** read-only

default replication factors for automatically created topics.

**delegation.token.expiry.time.ms**



**Type:** long

**Default:** 86400000

**Valid Values:** [1,...]

**Importance:** medium

**Dynamic update:** read-only

The token validity time in milliseconds before the token needs to be renewed. Default value 1 day.

### **delegation.token.master.key**

**Type:** password

**Default:** null

**Importance:** medium

**Dynamic update:** read-only

Master/secret key to generate and verify delegation tokens. Same key must be configured across all the brokers. If the key is not set or set to empty string, brokers will disable the delegation token support.

### **delegation.token.max.lifetime.ms**

**Type:** long

**Default:** 604800000

**Valid Values:** [1,...]

**Importance:** medium

**Dynamic update:** read-only

The token has a maximum lifetime beyond which it cannot be renewed anymore. Default value 7 days.

### **delete.records.purgatory.purge.interval.requests**

**Type:** int

**Default:** 1

**Importance:** medium

**Dynamic update:** read-only

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

### **fetch.purgatory.purge.interval.requests**

**Type:** int

**Default:** 1000

**Importance:** medium

**Dynamic update:** read-only

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

### **group.initial.rebalance.delay.ms**

**Type:** int

**Default:** 3000

**Importance:** medium

**Dynamic update:** read-only

The amount of time the group coordinator will wait for more consumers to join a new group before performing the first rebalance. A longer delay means potentially fewer rebalances, but increases the time until processing begins.

### **group.max.session.timeout.ms**

**Type:** int

**Default:** 1800000

**Importance:** medium

**Dynamic update:** read-only

The maximum allowed session timeout for registered consumers. Longer timeouts give consumers more time to process messages in between heartbeats at the cost of a longer time to detect failures.

**group.max.size**

**Type:** int

**Default:** 2147483647

**Valid Values:** [1,...]

**Importance:** medium

**Dynamic update:** read-only

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

**group.min.session.timeout.ms**

**Type:** int

**Default:** 6000

**Importance:** medium

**Dynamic update:** read-only

The minimum allowed session timeout for registered consumers. Shorter timeouts result in quicker failure detection at the cost of more frequent consumer heartbeating, which can overwhelm broker resources.

**inter.broker.listener.name**

**Type:** string

**Default:** null

**Importance:** medium

**Dynamic update:** read-only

Name of listener used for communication between brokers. If this is unset, the listener name is defined by `security.inter.broker.protocol`. It is an error to set this and `security.inter.broker.protocol` properties at the same time.

**inter.broker.protocol.version**

**Type:** string

**Default:** 2.3-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-IV0, 0.11.0-IV0, 0.11.0-IV1, 0.11.0-IV2, 1.0-IV0, 1.1-IV0, 2.0-IV0, 2.0-IV1, 2.1-IV0, 2.1-IV1, 2.1-IV2, 2.2-IV0, 2.2-IV1, 2.3-IV0, 2.3-IV1]

**Importance:** medium

**Dynamic update:** read-only

Specify which version of the inter-broker protocol will be used. This is typically bumped after all brokers were upgraded to a new version. Example 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 full list.

**log.cleaner.backoff.ms**

**Type:** long

**Default:** 15000

**Valid Values:** [0,...]

**Importance:** medium

**Dynamic update:** cluster-wide

The amount of time to sleep when there are no logs to clean.

**log.cleaner.dedupe.buffer.size**

**Type:** long

**Default:** 134217728

**Importance:** medium

**Dynamic update:** cluster-wide

The total memory used for log deduplication across all cleaner threads.

### **log.cleaner.delete.retention.ms**

**Type:** long

**Default:** 86400000

**Importance:** medium

**Dynamic update:** cluster-wide

How long are delete records retained?

### **log.cleaner.enable**

**Type:** boolean

**Default:** true

**Importance:** medium

**Dynamic update:** read-only

Enable the log cleaner process to run on the server. Should be enabled if using any topics with a `cleanup.policy=compact` including the internal offsets topic. If disabled those topics will not be compacted and continually grow in size.

### **log.cleaner.io.buffer.load.factor**

**Type:** double

**Default:** 0.9

**Importance:** medium

**Dynamic update:** cluster-wide

Log cleaner dedupe buffer load factor. The percentage full the dedupe buffer can become. A higher value will allow more log to be cleaned at once but will lead to more hash collisions.

### **log.cleaner.io.buffer.size**

**Type:** int

**Default:** 524288

**Valid Values:** [0,...]

**Importance:** medium

**Dynamic update:** cluster-wide

The total memory used for log cleaner I/O buffers across all cleaner threads.

### **log.cleaner.io.max.bytes.per.second**

**Type:** double

**Default:** 1.7976931348623157E308

**Importance:** medium

**Dynamic update:** cluster-wide

The log cleaner will be throttled so that the sum of its read and write i/o will be less than this value on average.

### **log.cleaner.max.compaction.lag.ms**

**Type:** long

**Default:** 9223372036854775807

**Importance:** medium

**Dynamic update:** cluster-wide

The maximum time a message will remain ineligible for compaction in the log. Only applicable for logs that are being compacted.

**log.cleaner.min.cleanable.ratio****Type:** double**Default:** 0.5**Importance:** medium**Dynamic update:** cluster-wide

The minimum ratio of dirty log to total log for a log to eligible for cleaning. If the `log.cleaner.max.compaction.lag.ms` or the `log.cleaner.min.compaction.lag.ms` configurations are also specified, then the log compactor considers the log eligible for compaction as soon as either: (i) the dirty ratio threshold has been met and the log has had dirty (uncompacted) records for at least the `log.cleaner.min.compaction.lag.ms` duration, or (ii) if the log has had dirty (uncompacted) records for at most the `log.cleaner.max.compaction.lag.ms` period.

**log.cleaner.min.compaction.lag.ms****Type:** long**Default:** 0**Importance:** medium**Dynamic update:** cluster-wide

The minimum time a message will remain uncompacted in the log. Only applicable for logs that are being compacted.

**log.cleaner.threads****Type:** int**Default:** 1**Valid Values:** [0,...]**Importance:** medium**Dynamic update:** cluster-wide

The number of background threads to use for log cleaning.

**log.cleanup.policy****Type:** list**Default:** delete**Valid Values:** [compact, delete]**Importance:** medium**Dynamic update:** cluster-wide

The default cleanup policy for segments beyond the retention window. A comma separated list of valid policies. Valid policies are: "delete" and "compact".

**log.index.interval.bytes****Type:** int**Default:** 4096**Valid Values:** [0,...]**Importance:** medium**Dynamic update:** cluster-wide

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

**log.index.size.max.bytes****Type:** int**Default:** 10485760**Valid Values:** [4,...]**Importance:** medium**Dynamic update:** cluster-wide

The maximum size in bytes of the offset index.

**log.message.format.version****Type:** string**Default:** 2.3-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-IV0, 0.11.0-IV0, 0.11.0-IV1, 0.11.0-IV2, 1.0-IV0, 1.1-IV0, 2.0-IV0, 2.0-IV1, 2.1-IV0, 2.1-IV1, 2.1-IV2, 2.2-IV0, 2.2-IV1, 2.3-IV0, 2.3-IV1]**Importance:** medium**Dynamic update:** read-only

Specify the message format version the broker will use to append messages to the logs. The value should be a valid `ApiVersion`. Some examples are: 0.8.2, 0.9.0.0, 0.10.0, check `ApiVersion` for more details. By setting a particular message format version, the user is certifying that all the existing messages on disk are smaller or equal than the specified version. Setting this value incorrectly will cause consumers with older versions to break as they will receive messages with a format that they don't understand.

**log.message.timestamp.difference.max.ms****Type:** long**Default:** 9223372036854775807**Importance:** medium**Dynamic update:** cluster-wide

The maximum difference allowed between the timestamp when a broker receives a message and the timestamp specified in the message. If `log.message.timestamp.type=CreateTime`, a message will be rejected if the difference in timestamp exceeds this threshold. This configuration is ignored if `log.message.timestamp.type=LogAppendTime`. The maximum timestamp difference allowed should be no greater than `log.retention.ms` to avoid unnecessarily frequent log rolling.

**log.message.timestamp.type****Type:** string**Default:** CreateTime**Valid Values:** [CreateTime, LogAppendTime]**Importance:** medium**Dynamic update:** cluster-wide

Define whether the timestamp in the message is message create time or log append time. The value should be either **CreateTime** or **LogAppendTime**.

**log.preallocate****Type:** boolean**Default:** false**Importance:** medium**Dynamic update:** cluster-wide

Should pre allocate file when create new segment? If you are using Kafka on Windows, you probably need to set it to true.

**log.retention.check.interval.ms****Type:** long**Default:** 300000**Valid Values:** [1,...]**Importance:** medium**Dynamic update:** read-only

The frequency in milliseconds that the log cleaner checks whether any log is eligible for deletion.

**max.connections**

**Type:** int

**Default:** 2147483647

**Valid Values:** [0,...]

**Importance:** medium

**Dynamic update:** cluster-wide

The maximum number of connections we allow in the broker at any time. This limit is applied in addition to any per-ip limits configured using `max.connections.per.ip`. Listener-level limits may also be configured by prefixing the config name with the listener prefix, for example,

**listener.name.internal.max.connections.** Broker-wide limit should be configured based on broker capacity while listener limits should be configured based on application requirements. New connections are blocked if either the listener or broker limit is reached. Connections on the inter-broker listener are permitted even if broker-wide limit is reached. The least recently used connection on another listener will be closed in this case.

### **max.connections.per.ip**

**Type:** int

**Default:** 2147483647

**Valid Values:** [0,...]

**Importance:** medium

**Dynamic update:** cluster-wide

The maximum number of connections we allow from each ip address. This can be set to 0 if there are overrides configured using `max.connections.per.ip.overrides` property. New connections from the ip address are dropped if the limit is reached.

### **max.connections.per.ip.overrides**

**Type:** string

**Default:** ""

**Importance:** medium

**Dynamic update:** cluster-wide

A comma-separated list of per-ip or hostname overrides to the default maximum number of connections. An example value is "hostName:100,127.0.0.1:200".

### **max.incremental.fetch.session.cache.slots**

**Type:** int

**Default:** 1000

**Valid Values:** [0,...]

**Importance:** medium

**Dynamic update:** read-only

The maximum number of incremental fetch sessions that we will maintain.

### **num.partitions**

**Type:** int

**Default:** 1

**Valid Values:** [1,...]

**Importance:** medium

**Dynamic update:** read-only

The default number of log partitions per topic.

### **password.encoder.old.secret**

**Type:** password

**Default:** null

**Importance:** medium

**Dynamic update:** read-only

The old secret that was used for encoding dynamically configured passwords. This is required only when the secret is updated. If specified, all dynamically encoded passwords are decoded using this old secret and re-encoded using `password.encoder.secret` when broker starts up.

### **password.encoder.secret**

**Type:** password

**Default:** null

**Importance:** medium

**Dynamic update:** read-only

The secret used for encoding dynamically configured passwords for this broker.

### **principal.builder.class**

**Type:** class

**Default:** null

**Importance:** medium

**Dynamic update:** per-broker

The fully qualified name of a class that implements the `KafkaPrincipalBuilder` interface, which is used to build the `KafkaPrincipal` object used during authorization. This config also supports the deprecated `PrincipalBuilder` interface which was previously used for client authentication over SSL. If no principal builder is defined, the default behavior depends on the security protocol in use. For SSL authentication, the principal will be derived using the rules defined by **`ssl.principal.mapping.rules`** applied on the distinguished name from the client certificate if one is provided; otherwise, if client authentication is not required, the principal name will be `ANONYMOUS`. For SASL authentication, the principal will be derived using the rules defined by **`sasl.kerberos.principal.to.local.rules`** if GSSAPI is in use, and the SASL authentication ID for other mechanisms. For `PLAINTEXT`, the principal will be `ANONYMOUS`.

### **producer.purgatory.purge.interval.requests**

**Type:** int

**Default:** 1000

**Importance:** medium

**Dynamic update:** read-only

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

### **queued.max.request.bytes**

**Type:** long

**Default:** -1

**Importance:** medium

**Dynamic update:** read-only

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

### **replica.fetch.backoff.ms**

**Type:** int

**Default:** 1000

**Valid Values:** [0,...]

**Importance:** medium

**Dynamic update:** read-only

The amount of time to sleep when fetch partition error occurs.

### **replica.fetch.max.bytes**

**Type:** int

**Default:** 1048576

**Valid Values:** [0,...]

**Importance:** medium

**Dynamic update:** read-only

The number of bytes of messages to attempt to fetch for each partition. This is not an absolute maximum, if the first record batch in the first non-empty partition of the fetch is larger than this value, the record batch will still be returned to ensure that progress can be made. The maximum record batch size accepted by the broker is defined via **message.max.bytes** (broker config) or **max.message.bytes** (topic config).

### **replica.fetch.response.max.bytes**

**Type:** int

**Default:** 10485760

**Valid Values:** [0,...]

**Importance:** medium

**Dynamic update:** read-only

Maximum bytes expected for the entire fetch response. Records are fetched in batches, and if the first record batch in the first non-empty partition of the fetch is larger than this value, the record batch will still be returned to ensure that progress can be made. As such, this is not an absolute maximum. The maximum record batch size accepted by the broker is defined via **message.max.bytes** (broker config) or **max.message.bytes** (topic config).

### **reserved.broker.max.id**

**Type:** int

**Default:** 1000

**Valid Values:** [0,...]

**Importance:** medium

**Dynamic update:** read-only

Max number that can be used for a broker.id.

### **sasl.client.callback.handler.class**

**Type:** class

**Default:** null

**Importance:** medium

**Dynamic update:** read-only

The fully qualified name of a SASL client callback handler class that implements the `AuthenticateCallbackHandler` interface.

### **sasl.enabled.mechanisms**

**Type:** list

**Default:** GSSAPI

**Importance:** medium

**Dynamic update:** per-broker

The list of SASL mechanisms enabled in the Kafka server. The list may contain any mechanism for which a security provider is available. Only GSSAPI is enabled by default.

### **sasl.jaas.config**

**Type:** password

**Default:** null

**Importance:** medium

**Dynamic update:** per-broker

JAAS login context parameters for SASL connections in the format used by JAAS configuration files. JAAS configuration file format is described [here](#). The format for the value is: `'loginModuleClass`



controlFlag (optionName=optionValue)\*;'. For brokers, the config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, listener.name.sasl\_ssl.scram-sha-256.sasl.jaas.config=com.example.ScramLoginModule required;

### **sasl.kerberos.kinit.cmd**

**Type:** string

**Default:** /usr/bin/kinit

**Importance:** medium

**Dynamic update:** per-broker

Kerberos kinit command path.

### **sasl.kerberos.min.time.before.relogin**

**Type:** long

**Default:** 60000

**Importance:** medium

**Dynamic update:** per-broker

Login thread sleep time between refresh attempts.

### **sasl.kerberos.principal.to.local.rules**

**Type:** list

**Default:** DEFAULT

**Importance:** medium

**Dynamic update:** per-broker

A list of rules for mapping from principal names to short names (typically operating system usernames). The rules are evaluated in order and the first rule that matches a principal name is used to map it to a short name. Any later rules in the list are ignored. By default, principal names of the form {username}/{hostname}@{REALM} are mapped to {username}. For more details on the format please see [security authorization and acls](#). Note that this configuration is ignored if an extension of KafkaPrincipalBuilder is provided by the **principal.builder.class** configuration.

### **sasl.kerberos.service.name**

**Type:** string

**Default:** null

**Importance:** medium

**Dynamic update:** per-broker

The Kerberos principal name that Kafka runs as. This can be defined either in Kafka's JAAS config or in Kafka's config.

### **sasl.kerberos.ticket.renew.jitter**

**Type:** double

**Default:** 0.05

**Importance:** medium

**Dynamic update:** per-broker

Percentage of random jitter added to the renewal time.

### **sasl.kerberos.ticket.renew.window.factor**

**Type:** double

**Default:** 0.8

**Importance:** medium

**Dynamic update:** per-broker

Login thread will sleep until the specified window factor of time from last refresh to ticket's expiry has been reached, at which time it will try to renew the ticket.

**sasl.login.callback.handler.class****Type:** class**Default:** null**Importance:** medium**Dynamic update:** read-only

The fully qualified name of a SASL login callback handler class that implements the `AuthenticateCallbackHandler` interface. For brokers, login callback handler config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.login.callback.handler.class=com.example.CustomScramLoginCallbackHandler`.

**sasl.login.class****Type:** class**Default:** null**Importance:** medium**Dynamic update:** read-only

The fully qualified name of a class that implements the `Login` interface. For brokers, login config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.login.class=com.example.CustomScramLogin`.

**sasl.login.refresh.buffer.seconds****Type:** short**Default:** 300**Importance:** medium**Dynamic update:** per-broker

The amount of buffer time before credential expiration to maintain when refreshing a credential, in seconds. If a refresh would otherwise occur closer to expiration than the number of buffer seconds then the refresh will be moved up to maintain as much of the buffer time as possible. Legal values are between 0 and 3600 (1 hour); a default value of 300 (5 minutes) is used if no value is specified. This value and `sasl.login.refresh.min.period.seconds` are both ignored if their sum exceeds the remaining lifetime of a credential. Currently applies only to `OAUTHBEARER`.

**sasl.login.refresh.min.period.seconds****Type:** short**Default:** 60**Importance:** medium**Dynamic update:** per-broker

The desired minimum time for the login refresh thread to wait before refreshing a credential, in seconds. Legal values are between 0 and 900 (15 minutes); a default value of 60 (1 minute) is used if no value is specified. This value and `sasl.login.refresh.buffer.seconds` are both ignored if their sum exceeds the remaining lifetime of a credential. Currently applies only to `OAUTHBEARER`.

**sasl.login.refresh.window.factor****Type:** double**Default:** 0.8**Importance:** medium**Dynamic update:** per-broker

Login refresh thread will sleep until the specified window factor relative to the credential's lifetime has been reached, at which time it will try to refresh the credential. Legal values are between 0.5 (50%) and 1.0 (100%) inclusive; a default value of 0.8 (80%) is used if no value is specified. Currently applies only to `OAUTHBEARER`.

**sasl.login.refresh.window.jitter**

**Type:** double

**Default:** 0.05

**Importance:** medium

**Dynamic update:** per-broker

The maximum amount of random jitter relative to the credential's lifetime that is added to the login refresh thread's sleep time. Legal values are between 0 and 0.25 (25%) inclusive; a default value of 0.05 (5%) is used if no value is specified. Currently applies only to OAUTHBEARER.

### **sasl.mechanism.inter.broker.protocol**

**Type:** string

**Default:** GSSAPI

**Importance:** medium

**Dynamic update:** per-broker

SASL mechanism used for inter-broker communication. Default is GSSAPI.

### **sasl.server.callback.handler.class**

**Type:** class

**Default:** null

**Importance:** medium

**Dynamic update:** read-only

The fully qualified name of a SASL server callback handler class that implements the AuthenticateCallbackHandler interface. Server callback handlers must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.plain.sasl.server.callback.handler.class=com.example.CustomPlainCallbackHandler`

### **security.inter.broker.protocol**

**Type:** string

**Default:** PLAINTEXT

**Importance:** medium

**Dynamic update:** read-only

Security protocol used to communicate between brokers. Valid values are: PLAINTEXT, SSL, SASL\_PLAINTEXT, SASL\_SSL. It is an error to set this and `inter.broker.listener.name` properties at the same time.

### **ssl.cipher.suites**

**Type:** list

**Default:** ""

**Importance:** medium

**Dynamic update:** per-broker

A list of cipher suites. This is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol. By default all the available cipher suites are supported.

### **ssl.client.auth**

**Type:** string

**Default:** none

**Valid Values:** [required, requested, none]

**Importance:** medium

**Dynamic update:** per-broker

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 option is set, the client can choose not to provide authentication information about itself.
- **ssl.client.auth=none** This means client authentication is not needed.

### **ssl.enabled.protocols**

**Type:** list

**Default:** TLSv1.2,TLSv1.1,TLSv1

**Importance:** medium

**Dynamic update:** per-broker

The list of protocols enabled for SSL connections.

### **ssl.key.password**

**Type:** password

**Default:** null

**Importance:** medium

**Dynamic update:** per-broker

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

### **ssl.keymanager.algorithm**

**Type:** string

**Default:** SunX509

**Importance:** medium

**Dynamic update:** per-broker

The algorithm used by the key manager factory for SSL connections. The default value is the key manager factory algorithm configured for the Java Virtual Machine.

### **ssl.keystore.location**

**Type:** string

**Default:** null

**Importance:** medium

**Dynamic update:** per-broker

The location of the key store file. This is optional for client and can be used for two-way authentication for client.

### **ssl.keystore.password**

**Type:** password

**Default:** null

**Importance:** medium

**Dynamic update:** per-broker

The store password for the key store file. This is optional for client and only needed if `ssl.keystore.location` is configured.

### **ssl.keystore.type**

**Type:** string

**Default:** JKS

**Importance:** medium

**Dynamic update:** per-broker

The file format of the key store file. This is optional for client.

### **ssl.protocol**

**Type:** string

**Default:** TLS

**Importance:** medium

**Dynamic update:** per-broker

The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for most cases. Allowed values in recent JVMs are TLS, TLSv1.1 and TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to known security vulnerabilities.

### **ssl.provider**

**Type:** string

**Default:** null

**Importance:** medium

**Dynamic update:** per-broker

The name of the security provider used for SSL connections. Default value is the default security provider of the JVM.

### **ssl.trustmanager.algorithm**

**Type:** string

**Default:** PKIX

**Importance:** medium

**Dynamic update:** per-broker

The algorithm used by trust manager factory for SSL connections. Default value is the trust manager factory algorithm configured for the Java Virtual Machine.

### **ssl.truststore.location**

**Type:** string

**Default:** null

**Importance:** medium

**Dynamic update:** per-broker

The location of the trust store file.

### **ssl.truststore.password**

**Type:** password

**Default:** null

**Importance:** medium

**Dynamic update:** per-broker

The password for the trust store file. If a password is not set access to the truststore is still available, but integrity checking is disabled.

### **ssl.truststore.type**

**Type:** string

**Default:** JKS

**Importance:** medium

**Dynamic update:** per-broker

The file format of the trust store file.

### **alter.config.policy.class.name**

**Type:** class

**Default:** null

**Importance:** low

**Dynamic update:** read-only

The alter configs policy class that should be used for validation. The class should implement the **org.apache.kafka.server.policy.AlterConfigPolicy** interface.

**alter.log.dirs.replication.quota.window.num**

**Type:** int

**Default:** 11

**Valid Values:** [1,...]

**Importance:** low

**Dynamic update:** read-only

The number of samples to retain in memory for alter log dirs replication quotas.

**alter.log.dirs.replication.quota.window.size.seconds**

**Type:** int

**Default:** 1

**Valid Values:** [1,...]

**Importance:** low

**Dynamic update:** read-only

The time span of each sample for alter log dirs replication quotas.

**authorizer.class.name**

**Type:** string

**Default:** ""

**Importance:** low

**Dynamic update:** read-only

The authorizer class that should be used for authorization.

**client.quota.callback.class**

**Type:** class

**Default:** null

**Importance:** low

**Dynamic update:** read-only

The fully qualified name of a class that implements the ClientQuotaCallback interface, which is used to determine quota limits applied to client requests. By default, <user, client-id>, <user> or <client-id> quotas stored in ZooKeeper are applied. For any given request, the most specific quota that matches the user principal of the session and the client-id of the request is applied.

**connection.failed.authentication.delay.ms**

**Type:** int

**Default:** 100

**Valid Values:** [0,...]

**Importance:** low

**Dynamic update:** read-only

Connection close delay on failed authentication: this is the time (in milliseconds) by which connection close will be delayed on authentication failure. This must be configured to be less than connections.max.idle.ms to prevent connection timeout.

**create.topic.policy.class.name**

**Type:** class

**Default:** null

**Importance:** low

**Dynamic update:** read-only

The create topic policy class that should be used for validation. The class should implement the **org.apache.kafka.server.policy.CreateTopicPolicy** interface.

### **delegation.token.expiry.check.interval.ms**

**Type:** long

**Default:** 3600000

**Valid Values:** [1,...]

**Importance:** low

**Dynamic update:** read-only

Scan interval to remove expired delegation tokens.

### **kafka.metrics.polling.interval.secs**

**Type:** int

**Default:** 10

**Valid Values:** [1,...]

**Importance:** low

**Dynamic update:** read-only

The metrics polling interval (in seconds) which can be used in `kafka.metrics.reporters` implementations.

### **kafka.metrics.reporters**

**Type:** list

**Default:** ""

**Importance:** low

**Dynamic update:** read-only

A list of classes to use as Yammer metrics custom reporters. The reporters should implement **kafka.metrics.KafkaMetricsReporter** trait. If a client wants to expose JMX operations on a custom reporter, the custom reporter needs to additionally implement an MBean trait that extends **kafka.metrics.KafkaMetricsReporterMBean** trait so that the registered MBean is compliant with the standard MBean convention.

### **listener.security.protocol.map**

**Type:** string

**Default:**

PLAINTEXT:PLAINTEXT,SSL:SSL,SASL\_PLAINTEXT:SASL\_PLAINTEXT,SASL\_SSL:SASL\_SSL

**Importance:** low

**Dynamic update:** per-broker

Map between listener names and security protocols. This must be defined for the same security protocol to be usable in more than one port or IP. For example, internal and external traffic can be separated even if SSL is required for both. Concretely, the user could define listeners with names INTERNAL and EXTERNAL and this property as: **INTERNAL:SSL,EXTERNAL:SSL**. As shown, key and value are separated by a colon and map entries are separated by commas. Each listener name should only appear once in the map. Different security (SSL and SASL) settings can be configured for each listener by adding a normalised prefix (the listener name is lowercased) to the config name. For example, to set a different keystore for the INTERNAL listener, a config with name **listener.name.internal.ssl.keystore.location** would be set. If the config for the listener name is not set, the config will fallback to the generic config (i.e. **ssl.keystore.location**).

### **log.message.downconversion.enable**

**Type:** boolean

**Default:** true

**Importance:** low

**Dynamic update:** cluster-wide

This configuration controls whether down-conversion of message formats is enabled to satisfy consume requests. When set to **false**, broker will not perform down-conversion for consumers expecting an older message format. The broker responds with **UNSUPPORTED\_VERSION** error for consume requests from such older clients. This configuration does not apply to any message format conversion that might be required for replication to followers.

### **metric.reporters**

**Type:** list

**Default:** ""

**Importance:** low

**Dynamic update:** cluster-wide

A list of classes to use as metrics reporters. Implementing the **org.apache.kafka.common.metrics.MetricsReporter** interface allows plugging in classes that will be notified of new metric creation. The JmxReporter is always included to register JMX statistics.

### **metrics.num.samples**

**Type:** int

**Default:** 2

**Valid Values:** [1,...]

**Importance:** low

**Dynamic update:** read-only

The number of samples maintained to compute metrics.

### **metrics.recording.level**

**Type:** string

**Default:** INFO

**Importance:** low

**Dynamic update:** read-only

The highest recording level for metrics.

### **metrics.sample.window.ms**

**Type:** long

**Default:** 30000

**Valid Values:** [1,...]

**Importance:** low

**Dynamic update:** read-only

The window of time a metrics sample is computed over.

### **password.encoder.cipher.algorithm**

**Type:** string

**Default:** AES/CBC/PKCS5Padding

**Importance:** low

**Dynamic update:** read-only

The Cipher algorithm used for encoding dynamically configured passwords.

### **password.encoder.iterations**

**Type:** int

**Default:** 4096

**Valid Values:** [1024,...]

**Importance:** low

**Dynamic update:** read-only

The iteration count used for encoding dynamically configured passwords.



**password.encoder.key.length**

**Type:** int

**Default:** 128

**Valid Values:** [8,...]

**Importance:** low

**Dynamic update:** read-only

The key length used for encoding dynamically configured passwords.

**password.encoder.keyfactory.algorithm**

**Type:** string

**Default:** null

**Importance:** low

**Dynamic update:** read-only

The SecretKeyFactory algorithm used for encoding dynamically configured passwords. Default is PBKDF2WithHmacSHA512 if available and PBKDF2WithHmacSHA1 otherwise.

**quota.window.num**

**Type:** int

**Default:** 11

**Valid Values:** [1,...]

**Importance:** low

**Dynamic update:** read-only

The number of samples to retain in memory for client quotas.

**quota.window.size.seconds**

**Type:** int

**Default:** 1

**Valid Values:** [1,...]

**Importance:** low

**Dynamic update:** read-only

The time span of each sample for client quotas.

**replication.quota.window.num**

**Type:** int

**Default:** 11

**Valid Values:** [1,...]

**Importance:** low

**Dynamic update:** read-only

The number of samples to retain in memory for replication quotas.

**replication.quota.window.size.seconds**

**Type:** int

**Default:** 1

**Valid Values:** [1,...]

**Importance:** low

**Dynamic update:** read-only

The time span of each sample for replication quotas.

**ssl.endpoint.identification.algorithm**

**Type:** string

**Default:** https

**Importance:** low

**Dynamic update:** per-broker

The endpoint identification algorithm to validate server hostname using server certificate.

### **ssl.principal.mapping.rules**

**Type:** list

**Default:** DEFAULT

**Importance:** low

**Dynamic update:** read-only

A list of rules for mapping from distinguished name from the client certificate to short name. The rules are evaluated in order and the first rule that matches a principal name is used to map it to a short name. Any later rules in the list are ignored. By default, distinguished name of the X.500 certificate will be the principal. For more details on the format please see [security authorization and acls](#). Note that this configuration is ignored if an extension of `KafkaPrincipalBuilder` is provided by the **principal.builder.class** configuration.

### **ssl.secure.random.implementation**

**Type:** string

**Default:** null

**Importance:** low

**Dynamic update:** per-broker

The SecureRandom PRNG implementation to use for SSL cryptography operations.

### **transaction.abort.timed.out.transaction.cleanup.interval.ms**

**Type:** int

**Default:** 60000

**Valid Values:** [1,...]

**Importance:** low

**Dynamic update:** read-only

The interval at which to rollback transactions that have timed out.

### **transaction.remove.expired.transaction.cleanup.interval.ms**

**Type:** int

**Default:** 3600000

**Valid Values:** [1,...]

**Importance:** low

**Dynamic update:** read-only

The interval at which to remove transactions that have expired due to **transactional.id.expiration.ms** passing.

### **zookeeper.sync.time.ms**

**Type:** int

**Default:** 2000

**Importance:** low

**Dynamic update:** read-only

How far a ZK follower can be behind a ZK leader.

## APPENDIX B. TOPIC CONFIGURATION PARAMETERS

### cleanup.policy

**Type:** list

**Default:** delete

**Valid Values:** [compact, delete]

**Server Default Property:** log.cleanup.policy

**Importance:** medium

A string that is either "delete" or "compact" or both. This string designates the retention policy to use on old log segments. The default policy ("delete") will discard old segments when their retention time or size limit has been reached. The "compact" setting will enable [log compaction](#) on the topic.

### compression.type

**Type:** string

**Default:** producer

**Valid Values:** [uncompressed, zstd, lz4, snappy, gzip, producer]

**Server Default Property:** compression.type

**Importance:** medium

Specify the final compression type for a given topic. This configuration accepts the standard compression codecs ('gzip', 'snappy', 'lz4', 'zstd'). It additionally accepts 'uncompressed' which is equivalent to no compression; and 'producer' which means retain the original compression codec set by the producer.

### delete.retention.ms

**Type:** long

**Default:** 86400000

**Valid Values:** [0,...]

**Server Default Property:** log.cleaner.delete.retention.ms

**Importance:** medium

The amount of time to retain delete tombstone markers for [log compacted](#) topics. This setting also gives a bound on the time in which a consumer must complete a read if they begin from offset 0 to ensure that they get a valid snapshot of the final stage (otherwise delete tombstones may be collected before they complete their scan).

### file.delete.delay.ms

**Type:** long

**Default:** 60000

**Valid Values:** [0,...]

**Server Default Property:** log.segment.delete.delay.ms

**Importance:** medium

The time to wait before deleting a file from the filesystem.

### flush.messages

**Type:** long

**Default:** 9223372036854775807

**Valid Values:** [0,...]

**Server Default Property:** log.flush.interval.messages

**Importance:** medium

This setting allows specifying an interval at which we will force an fsync of data written to the log. For example if this was set to 1 we would fsync after every message; if it were 5 we would fsync after every five messages. In general we recommend you not set this and use replication for durability and

allow the operating system's background flush capabilities as it is more efficient. This setting can be overridden on a per-topic basis (see [the per-topic configuration section](#)).

### **flush.ms**

**Type:** long

**Default:** 9223372036854775807

**Valid Values:** [0,...]

**Server Default Property:** log.flush.interval.ms

**Importance:** medium

This setting allows specifying a time interval at which we will force an fsync of data written to the log. For example if this was set to 1000 we would fsync after 1000 ms had passed. In general we recommend you not set this and use replication for durability and allow the operating system's background flush capabilities as it is more efficient.

### **follower.replication.throttled.replicas**

**Type:** list

**Default:** ""

**Valid Values:** [partitionId]:[brokerId],[partitionId]:[brokerId],...

**Server Default Property:** follower.replication.throttled.replicas

**Importance:** medium

A list of replicas for which log replication should be throttled on the follower side. The list should describe a set of replicas in the form [PartitionId]:[BrokerId],[PartitionId]:[BrokerId]:... or alternatively the wildcard '\*' can be used to throttle all replicas for this topic.

### **index.interval.bytes**

**Type:** int

**Default:** 4096

**Valid Values:** [0,...]

**Server Default Property:** log.index.interval.bytes

**Importance:** medium

This setting controls how frequently Kafka adds an index entry to its offset index. The default setting ensures that we index a message roughly every 4096 bytes. More indexing allows reads to jump closer to the exact position in the log but makes the index larger. You probably don't need to change this.

### **leader.replication.throttled.replicas**

**Type:** list

**Default:** ""

**Valid Values:** [partitionId]:[brokerId],[partitionId]:[brokerId],...

**Server Default Property:** leader.replication.throttled.replicas

**Importance:** medium

A list of replicas for which log replication should be throttled on the leader side. The list should describe a set of replicas in the form [PartitionId]:[BrokerId],[PartitionId]:[BrokerId]:... or alternatively the wildcard '\*' can be used to throttle all replicas for this topic.

### **max.compaction.lag.ms**

**Type:** long

**Default:** 9223372036854775807

**Valid Values:** [1,...]

**Server Default Property:** log.cleaner.max.compaction.lag.ms

**Importance:** medium

The maximum time a message will remain ineligible for compaction in the log. Only applicable for logs that are being compacted.

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

The largest record batch size allowed by Kafka. If this is increased and there are consumers older than 0.10.2, the consumers' fetch size must also be increased so that they can fetch record batches this large.

In the latest message format version, records are always grouped into batches for efficiency. In previous message format versions, uncompressed records are not grouped into batches and this limit only applies to a single record in that case.

**message.format.version****Type:** string**Default:** 2.3-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-IV0, 0.11.0-IV0, 0.11.0-IV1, 0.11.0-IV2, 1.0-IV0, 1.1-IV0, 2.0-IV0, 2.0-IV1, 2.1-IV0, 2.1-IV1, 2.1-IV2, 2.2-IV0, 2.2-IV1, 2.3-IV0, 2.3-IV1]**Server Default Property:** log.message.format.version**Importance:** medium

Specify the message format version the broker will use to append messages to the logs. The value should be a valid ApiVersion. Some examples are: 0.8.2, 0.9.0.0, 0.10.0, check ApiVersion for more details. By setting a particular message format version, the user is certifying that all the existing messages on disk are smaller or equal than the specified version. Setting this value incorrectly will cause consumers with older versions to break as they will receive messages with a format that they don't understand.

**message.timestamp.difference.max.ms****Type:** long**Default:** 9223372036854775807**Valid Values:** [0,...]**Server Default Property:** log.message.timestamp.difference.max.ms**Importance:** medium

The maximum difference allowed between the timestamp when a broker receives a message and the timestamp specified in the message. If message.timestamp.type=CreateTime, a message will be rejected if the difference in timestamp exceeds this threshold. This configuration is ignored if message.timestamp.type=LogAppendTime.

**message.timestamp.type****Type:** string**Default:** CreateTime**Valid Values:** [CreateTime, LogAppendTime]**Server Default Property:** log.message.timestamp.type**Importance:** medium

Define whether the timestamp in the message is message create time or log append time. The value should be either **CreateTime** or **LogAppendTime**.

**min.cleanable.dirty.ratio****Type:** double**Default:** 0.5**Valid Values:** [0,...,1]

**Server Default Property:** log.cleaner.min.cleanable.ratio

**Importance:** medium

This configuration controls how frequently the log compactor will attempt to clean the log (assuming [log compaction](#) is enabled). By default we will avoid cleaning a log where more than 50% of the log has been compacted. This ratio bounds the maximum space wasted in the log by duplicates (at 50% at most 50% of the log could be duplicates). A higher ratio will mean fewer, more efficient cleanings but will mean more wasted space in the log. If the `max.compaction.lag.ms` or the `min.compaction.lag.ms` configurations are also specified, then the log compactor considers the log eligible for compaction as soon as either: (i) the dirty ratio threshold has been met and the log has had dirty (uncompacted) records for at least the `min.compaction.lag.ms` duration, or (ii) if the log has had dirty (uncompacted) records for at most the `max.compaction.lag.ms` period.

### **min.compaction.lag.ms**

**Type:** long

**Default:** 0

**Valid Values:** [0,...]

**Server Default Property:** log.cleaner.min.compaction.lag.ms

**Importance:** medium

The minimum time a message will remain uncompacted in the log. Only applicable for logs that are being compacted.

### **min.insync.replicas**

**Type:** int

**Default:** 1

**Valid Values:** [1,...]

**Server Default Property:** min.insync.replicas

**Importance:** medium

When a producer sets `acks` to "all" (or "-1"), this configuration specifies the minimum number of replicas that must acknowledge a write for the write to be considered successful. If this minimum cannot be met, then the producer will raise an exception (either `NotEnoughReplicas` or `NotEnoughReplicasAfterAppend`). When used together, **min.insync.replicas** and **acks** allow you to enforce greater durability guarantees. A typical scenario would be to create a topic with a replication factor of 3, set **min.insync.replicas** to 2, and produce with **acks** of "all". This will ensure that the producer raises an exception if a majority of replicas do not receive a write.

### **preallocate**

**Type:** boolean

**Default:** false

**Server Default Property:** log.preallocate

**Importance:** medium

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

### **retention.bytes**

**Type:** long

**Default:** -1

**Server Default Property:** log.retention.bytes

**Importance:** medium

This configuration controls the maximum size a partition (which consists of log segments) can grow to before we will discard old log segments to free up space if we are using the "delete" retention policy. By default there is no size limit only a time limit. Since this limit is enforced at the partition level, multiply it by the number of partitions to compute the topic retention in bytes.

### **retention.ms**

**Type:** long

**Default:** 604800000

**Valid Values:** [-1,...]

**Server Default Property:** log.retention.ms

**Importance:** medium

This configuration controls the maximum time we will retain a log before we will discard old log segments to free up space if we are using the "delete" retention policy. This represents an SLA on how soon consumers must read their data. If set to -1, no time limit is applied.

### **segment.bytes**

**Type:** int

**Default:** 1073741824

**Valid Values:** [14,...]

**Server Default Property:** log.segment.bytes

**Importance:** medium

This configuration controls the segment file size for the log. Retention and cleaning is always done a file at a time so a larger segment size means fewer files but less granular control over retention.

### **segment.index.bytes**

**Type:** int

**Default:** 10485760

**Valid Values:** [0,...]

**Server Default Property:** log.index.size.max.bytes

**Importance:** medium

This configuration controls the size of the index that maps offsets to file positions. We preallocate this index file and shrink it only after log rolls. You generally should not need to change this setting.

### **segment.jitter.ms**

**Type:** long

**Default:** 0

**Valid Values:** [0,...]

**Server Default Property:** log.roll.jitter.ms

**Importance:** medium

The maximum random jitter subtracted from the scheduled segment roll time to avoid thundering herds of segment rolling.

### **segment.ms**

**Type:** long

**Default:** 604800000

**Valid Values:** [1,...]

**Server Default Property:** log.roll.ms

**Importance:** medium

This configuration controls the period of time after which Kafka will force the log to roll even if the segment file isn't full to ensure that retention can delete or compact old data.

### **unclean.leader.election.enable**

**Type:** boolean

**Default:** false

**Server Default Property:** unclean.leader.election.enable

**Importance:** medium

Indicates whether to enable replicas not in the ISR set to be elected as leader as a last resort, even though doing so may result in data loss.

**message.downconversion.enable****Type:** boolean**Default:** true**Server Default Property:** log.message.downconversion.enable**Importance:** low

This configuration controls whether down-conversion of message formats is enabled to satisfy consume requests. When set to **false**, broker will not perform down-conversion for consumers expecting an older message format. The broker responds with **UNSUPPORTED\_VERSION** error for consume requests from such older clients. This configuration does not apply to any message format conversion that might be required for replication to followers.



## APPENDIX C. CONSUMER CONFIGURATION PARAMETERS

### key.deserializer

**Type:** class

**Importance:** high

Deserializer class for key that implements the

**org.apache.kafka.common.serialization.Deserializer** interface.

### value.deserializer

**Type:** class

**Importance:** high

Deserializer class for value that implements the

**org.apache.kafka.common.serialization.Deserializer** interface.

### bootstrap.servers

**Type:** list

**Default:** ""

**Valid Values:** non-null string

**Importance:** high

A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. The client will make use of all servers irrespective of which servers are specified here for bootstrapping—this list only impacts the initial hosts used to discover the full set of servers. This list should be in the form **host1:port1,host2:port2,...**. Since these servers are just used for the initial connection to discover the full cluster membership (which may change dynamically), this list need not contain the full set of servers (you may want more than one, though, in case a server is down).

### fetch.min.bytes

**Type:** int

**Default:** 1

**Valid Values:** [0,...]

**Importance:** high

The minimum amount of data the server should return for a fetch request. If insufficient data is available the request will wait for that much data to accumulate before answering the request. The default setting of 1 byte means that fetch requests are answered as soon as a single byte of data is available or the fetch request times out waiting for data to arrive. Setting this to something greater than 1 will cause the server to wait for larger amounts of data to accumulate which can improve server throughput a bit at the cost of some additional latency.

### group.id

**Type:** string

**Default:** null

**Importance:** high

A unique string that identifies the consumer group this consumer belongs to. This property is required if the consumer uses either the group management functionality by using **subscribe(topic)** or the Kafka-based offset management strategy.

### heartbeat.interval.ms

**Type:** int

**Default:** 3000

**Importance:** high

The expected time between heartbeats to the consumer coordinator when using Kafka's group management facilities. Heartbeats are used to ensure that the consumer's session stays active and to

facilitate rebalancing when new consumers join or leave the group. The value must be set lower than **session.timeout.ms**, but typically should be set no higher than 1/3 of that value. It can be adjusted even lower to control the expected time for normal rebalances.

### **max.partition.fetch.bytes**

**Type:** int

**Default:** 1048576

**Valid Values:** [0,...]

**Importance:** high

The maximum amount of data per-partition the server will return. Records are fetched in batches by the consumer. If the first record batch in the first non-empty partition of the fetch is larger than this limit, the batch will still be returned to ensure that the consumer can make progress. The maximum record batch size accepted by the broker is defined via **message.max.bytes** (broker config) or **max.message.bytes** (topic config). See `fetch.max.bytes` for limiting the consumer request size.

### **session.timeout.ms**

**Type:** int

**Default:** 10000

**Importance:** high

The timeout used to detect consumer failures when using Kafka's group management facility. The consumer sends periodic heartbeats to indicate its liveness to the broker. If no heartbeats are received by the broker before the expiration of this session timeout, then the broker will remove this consumer from the group and initiate a rebalance. Note that the value must be in the allowable range as configured in the broker configuration by **group.min.session.timeout.ms** and **group.max.session.timeout.ms**.

### **ssl.key.password**

**Type:** password

**Default:** null

**Importance:** high

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

### **ssl.keystore.location**

**Type:** string

**Default:** null

**Importance:** high

The location of the key store file. This is optional for client and can be used for two-way authentication for client.

### **ssl.keystore.password**

**Type:** password

**Default:** null

**Importance:** high

The store password for the key store file. This is optional for client and only needed if `ssl.keystore.location` is configured.

### **ssl.truststore.location**

**Type:** string

**Default:** null

**Importance:** high

The location of the trust store file.

**ssl.truststore.password****Type:** password**Default:** null**Importance:** high

The password for the trust store file. If a password is not set access to the truststore is still available, but integrity checking is disabled.

**allow.auto.create.topics****Type:** boolean**Default:** true**Importance:** medium

Allow automatic topic creation on the broker when subscribing to or assigning a topic. A topic being subscribed to will be automatically created only if the broker allows for it using **auto.create.topics.enable** broker configuration. This configuration must be set to **false** when using brokers older than 0.11.0.

**auto.offset.reset****Type:** string**Default:** latest**Valid Values:** [latest, earliest, none]**Importance:** medium

What to do when there is no initial offset in Kafka or if the current offset does not exist any more on the server (e.g. because that data has been deleted):

- earliest: automatically reset the offset to the earliest offset
- latest: automatically reset the offset to the latest offset
- none: throw exception to the consumer if no previous offset is found for the consumer's group
- anything else: throw exception to the consumer.

**client.dns.lookup****Type:** string**Default:** default**Valid Values:** [default, use\_all\_dns\_ips, resolve\_canonical\_bootstrap\_servers\_only]**Importance:** medium

Controls how the client uses DNS lookups.

If set to **use\_all\_dns\_ips** then, when the lookup returns multiple IP addresses for a hostname, they will all be attempted to connect to before failing the connection. Applies to both bootstrap and advertised servers.

If the value is **resolve\_canonical\_bootstrap\_servers\_only** each entry will be resolved and expanded into a list of canonical names.

**connections.max.idle.ms****Type:** long**Default:** 540000**Importance:** medium

Close idle connections after the number of milliseconds specified by this config.

**default.api.timeout.ms****Type:** int**Default:** 60000**Valid Values:** [0,...]**Importance:** medium

Specifies the timeout (in milliseconds) for consumer APIs that could block. This configuration is used as the default timeout for all consumer operations that do not explicitly accept a **timeout** parameter.

**enable.auto.commit****Type:** boolean**Default:** true**Importance:** medium

If true the consumer's offset will be periodically committed in the background.

**exclude.internal.topics****Type:** boolean**Default:** true**Importance:** medium

Whether internal topics matching a subscribed pattern should be excluded from the subscription. It is always possible to explicitly subscribe to an internal topic.

**fetch.max.bytes****Type:** int**Default:** 52428800**Valid Values:** [0,...]**Importance:** medium

The maximum amount of data the server should return for a fetch request. Records are fetched in batches by the consumer, and if the first record batch in the first non-empty partition of the fetch is larger than this value, the record batch will still be returned to ensure that the consumer can make progress. As such, this is not an absolute maximum. The maximum record batch size accepted by the broker is defined via **message.max.bytes** (broker config) or **max.message.bytes** (topic config). Note that the consumer performs multiple fetches in parallel.

**group.instance.id****Type:** string**Default:** null**Importance:** medium

A unique identifier of the consumer instance provided by end user. Only non-empty strings are permitted. If set, the consumer is treated as a static member, which means that only one instance with this ID is allowed in the consumer group at any time. This can be used in combination with a larger session timeout to avoid group rebalances caused by transient unavailability (e.g. process restarts). If not set, the consumer will join the group as a dynamic member, which is the traditional behavior.

**isolation.level****Type:** string**Default:** read\_uncommitted**Valid Values:** [read\_committed, read\_uncommitted]**Importance:** medium

Controls how to read messages written transactionally. If set to **read\_committed**, `consumer.poll()` will only return transactional messages which have been committed. If set to `read_uncommitted` (the default), `consumer.poll()` will return all messages, even transactional messages which have been

aborted. Non-transactional messages will be returned unconditionally in either mode.

Messages will always be returned in offset order. Hence, in **read\_committed** mode, `consumer.poll()` will only return messages up to the last stable offset (LSO), which is the one less than the offset of the first open transaction. In particular any messages appearing after messages belonging to ongoing transactions will be withheld until the relevant transaction has been completed. As a result, **read\_committed** consumers will not be able to read up to the high watermark when there are in flight transactions.

Further, when in `read_committed` the `seekToEnd` method will return the LSO.

### **max.poll.interval.ms**

**Type:** int

**Default:** 300000

**Valid Values:** [1,...]

**Importance:** medium

The maximum delay between invocations of `poll()` when using consumer group management. This places an upper bound on the amount of time that the consumer can be idle before fetching more records. If `poll()` is not called before expiration of this timeout, then the consumer is considered failed and the group will rebalance in order to reassign the partitions to another member.

### **max.poll.records**

**Type:** int

**Default:** 500

**Valid Values:** [1,...]

**Importance:** medium

The maximum number of records returned in a single call to `poll()`.

### **partition.assignment.strategy**

**Type:** list

**Default:** class org.apache.kafka.clients.consumer.RangeAssignor

**Valid Values:** non-null string

**Importance:** medium

The class name of the partition assignment strategy that the client will use to distribute partition ownership amongst consumer instances when group management is used.

### **receive.buffer.bytes**

**Type:** int

**Default:** 65536

**Valid Values:** [-1,...]

**Importance:** medium

The size of the TCP receive buffer (SO\_RCVBUF) to use when reading data. If the value is -1, the OS default will be used.

### **request.timeout.ms**

**Type:** int

**Default:** 30000

**Valid Values:** [0,...]

**Importance:** medium

The configuration controls the maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.

**sasl.client.callback.handler.class****Type:** class**Default:** null**Importance:** medium

The fully qualified name of a SASL client callback handler class that implements the `AuthenticateCallbackHandler` interface.

**sasl.jaas.config****Type:** password**Default:** null**Importance:** medium

JAAS login context parameters for SASL connections in the format used by JAAS configuration files. JAAS configuration file format is described [here](#). The format for the value is: 'loginModuleClass controlFlag (optionName=optionValue)\*;'. For brokers, the config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.jaas.config=com.example.ScramLoginModule required;`

**sasl.kerberos.service.name****Type:** string**Default:** null**Importance:** medium

The Kerberos principal name that Kafka runs as. This can be defined either in Kafka's JAAS config or in Kafka's config.

**sasl.login.callback.handler.class****Type:** class**Default:** null**Importance:** medium

The fully qualified name of a SASL login callback handler class that implements the `AuthenticateCallbackHandler` interface. For brokers, login callback handler config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.login.callback.handler.class=com.example.CustomScramLoginCallbackHandler.`

**sasl.login.class****Type:** class**Default:** null**Importance:** medium

The fully qualified name of a class that implements the `Login` interface. For brokers, login config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.login.class=com.example.CustomScramLogin.`

**sasl.mechanism****Type:** string**Default:** GSSAPI**Importance:** medium

SASL mechanism used for client connections. This may be any mechanism for which a security provider is available. GSSAPI is the default mechanism.

**security.protocol**

**Type:** string

**Default:** PLAINTEXT

**Importance:** medium

Protocol used to communicate with brokers. Valid values are: PLAINTEXT, SSL, SASL\_PLAINTEXT, SASL\_SSL.

### **send.buffer.bytes**

**Type:** int

**Default:** 131072

**Valid Values:** [-1,...]

**Importance:** medium

The size of the TCP send buffer (SO\_SNDBUF) to use when sending data. If the value is -1, the OS default will be used.

### **ssl.enabled.protocols**

**Type:** list

**Default:** TLSv1.2,TLSv1.1,TLSv1

**Importance:** medium

The list of protocols enabled for SSL connections.

### **ssl.keystore.type**

**Type:** string

**Default:** JKS

**Importance:** medium

The file format of the key store file. This is optional for client.

### **ssl.protocol**

**Type:** string

**Default:** TLS

**Importance:** medium

The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for most cases. Allowed values in recent JVMs are TLS, TLSv1.1 and TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to known security vulnerabilities.

### **ssl.provider**

**Type:** string

**Default:** null

**Importance:** medium

The name of the security provider used for SSL connections. Default value is the default security provider of the JVM.

### **ssl.truststore.type**

**Type:** string

**Default:** JKS

**Importance:** medium

The file format of the trust store file.

### **auto.commit.interval.ms**

**Type:** int

**Default:** 5000

**Valid Values:** [0,...]

**Importance:** low

The frequency in milliseconds that the consumer offsets are auto-committed to Kafka if **enable.auto.commit** is set to **true**.

### **check.crcs**

**Type:** boolean

**Default:** true

**Importance:** low

Automatically check the CRC32 of the records consumed. This ensures no on-the-wire or on-disk corruption to the messages occurred. This check adds some overhead, so it may be disabled in cases seeking extreme performance.

### **client.id**

**Type:** string

**Default:** ""

**Importance:** low

An id string to pass to the server when making requests. The purpose of this is to be able to track the source of requests beyond just ip/port by allowing a logical application name to be included in server-side request logging.

### **client.rack**

**Type:** string

**Default:** ""

**Importance:** low

A rack identifier for this client. This can be any string value which indicates where this client is physically located. It corresponds with the broker config 'broker.rack'.

### **fetch.max.wait.ms**

**Type:** int

**Default:** 500

**Valid Values:** [0,...]

**Importance:** low

The maximum amount of time the server will block before answering the fetch request if there isn't sufficient data to immediately satisfy the requirement given by `fetch.min.bytes`.

### **interceptor.classes**

**Type:** list

**Default:** ""

**Valid Values:** non-null string

**Importance:** low

A list of classes to use as interceptors. Implementing the **org.apache.kafka.clients.consumer.ConsumerInterceptor** interface allows you to intercept (and possibly mutate) records received by the consumer. By default, there are no interceptors.

### **metadata.max.age.ms**

**Type:** long

**Default:** 300000

**Valid Values:** [0,...]

**Importance:** low

The period of time in milliseconds after which we force a refresh of metadata even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions.



**metric.reporters**

**Type:** list

**Default:** ""

**Valid Values:** non-null string

**Importance:** low

A list of classes to use as metrics reporters. Implementing the

**org.apache.kafka.common.metrics.MetricsReporter** interface allows plugging in classes that will be notified of new metric creation. The JmxReporter is always included to register JMX statistics.

**metrics.num.samples**

**Type:** int

**Default:** 2

**Valid Values:** [1,...]

**Importance:** low

The number of samples maintained to compute metrics.

**metrics.recording.level**

**Type:** string

**Default:** INFO

**Valid Values:** [INFO, DEBUG]

**Importance:** low

The highest recording level for metrics.

**metrics.sample.window.ms**

**Type:** long

**Default:** 30000

**Valid Values:** [0,...]

**Importance:** low

The window of time a metrics sample is computed over.

**reconnect.backoff.max.ms**

**Type:** long

**Default:** 1000

**Valid Values:** [0,...]

**Importance:** low

The maximum amount of time in milliseconds to wait when reconnecting to a broker that has repeatedly failed to connect. If provided, the backoff per host will increase exponentially for each consecutive connection failure, up to this maximum. After calculating the backoff increase, 20% random jitter is added to avoid connection storms.

**reconnect.backoff.ms**

**Type:** long

**Default:** 50

**Valid Values:** [0,...]

**Importance:** low

The base amount of time to wait before attempting to reconnect to a given host. This avoids repeatedly connecting to a host in a tight loop. This backoff applies to all connection attempts by the client to a broker.

**retry.backoff.ms**

**Type:** long

**Default:** 100

**Valid Values:** [0,...]

**Importance:** low

The amount of time to wait before attempting to retry a failed request to a given topic partition. This avoids repeatedly sending requests in a tight loop under some failure scenarios.

#### **sasl.kerberos.kinit.cmd**

**Type:** string

**Default:** /usr/bin/kinit

**Importance:** low

Kerberos kinit command path.

#### **sasl.kerberos.min.time.before.relogin**

**Type:** long

**Default:** 60000

**Importance:** low

Login thread sleep time between refresh attempts.

#### **sasl.kerberos.ticket.renew.jitter**

**Type:** double

**Default:** 0.05

**Importance:** low

Percentage of random jitter added to the renewal time.

#### **sasl.kerberos.ticket.renew.window.factor**

**Type:** double

**Default:** 0.8

**Importance:** low

Login thread will sleep until the specified window factor of time from last refresh to ticket's expiry has been reached, at which time it will try to renew the ticket.

#### **sasl.login.refresh.buffer.seconds**

**Type:** short

**Default:** 300

**Valid Values:** [0,...,3600]

**Importance:** low

The amount of buffer time before credential expiration to maintain when refreshing a credential, in seconds. If a refresh would otherwise occur closer to expiration than the number of buffer seconds then the refresh will be moved up to maintain as much of the buffer time as possible. Legal values are between 0 and 3600 (1 hour); a default value of 300 (5 minutes) is used if no value is specified. This value and `sasl.login.refresh.min.period.seconds` are both ignored if their sum exceeds the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

#### **sasl.login.refresh.min.period.seconds**

**Type:** short

**Default:** 60

**Valid Values:** [0,...,900]

**Importance:** low

The desired minimum time for the login refresh thread to wait before refreshing a credential, in seconds. Legal values are between 0 and 900 (15 minutes); a default value of 60 (1 minute) is used if no value is specified. This value and `sasl.login.refresh.buffer.seconds` are both ignored if their sum exceeds the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

**sasl.login.refresh.window.factor**

**Type:** double

**Default:** 0.8

**Valid Values:** [0.5,...,1.0]

**Importance:** low

Login refresh thread will sleep until the specified window factor relative to the credential's lifetime has been reached, at which time it will try to refresh the credential. Legal values are between 0.5 (50%) and 1.0 (100%) inclusive; a default value of 0.8 (80%) is used if no value is specified. Currently applies only to OAUTHBEARER.

**sasl.login.refresh.window.jitter**

**Type:** double

**Default:** 0.05

**Valid Values:** [0.0,...,0.25]

**Importance:** low

The maximum amount of random jitter relative to the credential's lifetime that is added to the login refresh thread's sleep time. Legal values are between 0 and 0.25 (25%) inclusive; a default value of 0.05 (5%) is used if no value is specified. Currently applies only to OAUTHBEARER.

**ssl.cipher.suites**

**Type:** list

**Default:** null

**Importance:** low

A list of cipher suites. This is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol. By default all the available cipher suites are supported.

**ssl.endpoint.identification.algorithm**

**Type:** string

**Default:** https

**Importance:** low

The endpoint identification algorithm to validate server hostname using server certificate.

**ssl.keymanager.algorithm**

**Type:** string

**Default:** SunX509

**Importance:** low

The algorithm used by key manager factory for SSL connections. Default value is the key manager factory algorithm configured for the Java Virtual Machine.

**ssl.secure.random.implementation**

**Type:** string

**Default:** null

**Importance:** low

The SecureRandom PRNG implementation to use for SSL cryptography operations.

**ssl.trustmanager.algorithm**

**Type:** string

**Default:** PKIX

**Importance:** low

The algorithm used by trust manager factory for SSL connections. Default value is the trust manager factory algorithm configured for the Java Virtual Machine.

## APPENDIX D. PRODUCER CONFIGURATION PARAMETERS

### key.serializer

**Type:** class

**Importance:** high

Serializer class for key that implements the **org.apache.kafka.common.serialization.Serializer** interface.

### value.serializer

**Type:** class

**Importance:** high

Serializer class for value that implements the **org.apache.kafka.common.serialization.Serializer** interface.

### acks

**Type:** string

**Default:** 1

**Valid Values:** [all, -1, 0, 1]

**Importance:** high

The number of acknowledgments the producer requires the leader to have received before considering a request complete. This controls the durability of records that are 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 record will be immediately added to the socket buffer and considered sent. No guarantee can be made that the server has received the record in this case, and the **retries** configuration will not take effect (as the client won't 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 full acknowledgement from all followers. In this case should the leader fail immediately after acknowledging the record but before the followers have replicated it then the record will be lost.
- **acks=all** This means the leader will wait for the full set of in-sync replicas to acknowledge the record. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee. This is equivalent to the **acks=-1** setting.

### bootstrap.servers

**Type:** list

**Default:** ""

**Valid Values:** non-null string

**Importance:** high

A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. The client will make use of all servers irrespective of which servers are specified here for bootstrapping—this list only impacts the initial hosts used to discover the full set of servers. This list should be in the form **host1:port1,host2:port2,...** Since these servers are just used for the initial connection to discover the full cluster membership (which may change dynamically), this list need not contain the full set of servers (you may want more than one, though, in case a server is down).

### buffer.memory

**Type:** long

**Default:** 33554432

**Valid Values:** [0,...]

**Importance:** high

The total bytes of memory the producer can use to buffer records waiting to be sent to the server. If records are sent faster than they can be delivered to the 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 since not all memory the producer uses is used for buffering. Some additional memory will be used for compression (if compression is enabled) as well as for maintaining in-flight requests.

### **compression.type**

**Type:** string

**Default:** none

**Importance:** high

The compression type for all data generated by the producer. The default is none (i.e. no compression). Valid values are **none**, **gzip**, **snappy**, **lz4**, or **zstd**. Compression is of full batches of data, so the efficacy of batching will also impact the compression ratio (more batching means better compression).

### **retries**

**Type:** int

**Default:** 2147483647

**Valid Values:** [0,...,2147483647]

**Importance:** high

Setting a value greater than zero will cause the client to resend any record whose send fails with a potentially transient error. Note that this retry is no different than if the client resent the record upon receiving the error. Allowing retries without setting **max.in.flight.requests.per.connection** to 1 will potentially change the ordering of records because if two batches are sent to a single partition, and the first fails and is retried but the second succeeds, then the records in the second batch may appear first. Note additionally that produce requests will be failed before the number of retries has been exhausted if the timeout configured by **delivery.timeout.ms** expires first before successful acknowledgement. Users should generally prefer to leave this config unset and instead use **delivery.timeout.ms** to control retry behavior.

### **ssl.key.password**

**Type:** password

**Default:** null

**Importance:** high

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

### **ssl.keystore.location**

**Type:** string

**Default:** null

**Importance:** high

The location of the key store file. This is optional for client and can be used for two-way authentication for client.

### **ssl.keystore.password**

**Type:** password

**Default:** null

**Importance:** high

The store password for the key store file. This is optional for client and only needed if `ssl.keystore.location` is configured.

### **ssl.truststore.location**

**Type:** string

**Default:** null

**Importance:** high

The location of the trust store file.

### **ssl.truststore.password**

**Type:** password

**Default:** null

**Importance:** high

The password for the trust store file. If a password is not set access to the truststore is still available, but integrity checking is disabled.

### **batch.size**

**Type:** int

**Default:** 16384

**Valid Values:** [0,...]

**Importance:** medium

The producer will attempt to batch records together into fewer requests whenever multiple records are being sent to the same partition. This helps performance on 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 disable batching entirely). A very large batch size may use memory a bit more wastefully as we will always allocate a buffer of the specified batch size in anticipation of additional records.

### **client.dns.lookup**

**Type:** string

**Default:** default

**Valid Values:** [default, use\_all\_dns\_ips, resolve\_canonical\_bootstrap\_servers\_only]

**Importance:** medium

Controls how the client uses DNS lookups.

If set to **use\_all\_dns\_ips** then, when the lookup returns multiple IP addresses for a hostname, they will all be attempted to connect to before failing the connection. Applies to both bootstrap and advertised servers.

If the value is **resolve\_canonical\_bootstrap\_servers\_only** each entry will be resolved and expanded into a list of canonical names.

### **client.id**

**Type:** string

**Default:** ""

**Importance:** medium

An id string to pass to the server when making requests. The purpose of this is to be able to track the source of requests beyond just ip/port by allowing a logical application name to be included in server-side request logging.

### **connections.max.idle.ms**

**Type:** long

**Default:** 540000

**Importance:** medium

Close idle connections after the number of milliseconds specified by this config.

### **delivery.timeout.ms**

**Type:** int

**Default:** 120000

**Valid Values:** [0,...]

**Importance:** medium

An upper bound on the time to report success or failure after a call to **send()** returns. This limits the total time that a record will be delayed prior to sending, the time to await acknowledgement from the broker (if expected), and the time allowed for retrievable send failures. The producer may report failure to send a record earlier than this config if either an unrecoverable error is encountered, the retries have been exhausted, or the record is added to a batch which reached an earlier delivery expiration deadline. The value of this config should be greater than or equal to the sum of **request.timeout.ms** and **linger.ms**.

### **linger.ms**

**Type:** long

**Default:** 0

**Valid Values:** [0,...]

**Importance:** medium

The producer groups together any records that arrive in between request transmissions into a single batched request. Normally this occurs only under load when records arrive faster than they can be sent out. However in some circumstances the client may want to reduce the number of requests even under moderate load. This setting accomplishes this by adding a small amount of artificial delay—that is, rather than immediately sending out a record the producer will wait for up to the given delay to allow other records to be sent so that the sends can be batched together. This can be thought of as analogous to Nagle's algorithm in TCP. This setting gives the upper bound on the delay for batching: once we get **batch.size** worth of records for a partition it will be sent immediately regardless of this setting, however if we have fewer than this many bytes accumulated for this partition we will 'linger' for the specified time waiting for more records to show up. This setting defaults to 0 (i.e. no delay). Setting **linger.ms=5**, for example, would have the effect of reducing the number of requests sent but would add up to 5ms of latency to records sent in the absence of load.

### **max.block.ms**

**Type:** long

**Default:** 60000

**Valid Values:** [0,...]

**Importance:** medium

The configuration controls how long **KafkaProducer.send()** and **KafkaProducer.partitionsFor()** will block. These methods can be blocked either because the buffer is full or metadata unavailable. Blocking in the user-supplied serializers or partitioner will not be counted against this timeout.

### **max.request.size**

**Type:** int



**Default:** 1048576

**Valid Values:** [0,...]

**Importance:** medium

The maximum size of a request in bytes. This setting will limit the number of record batches the producer will send in a single request to avoid sending huge requests. This is also effectively a cap on the maximum record batch size. Note that the server has its own cap on record batch size which may be different from this.

### **partitioner.class**

**Type:** class

**Default:** org.apache.kafka.clients.producer.internals.DefaultPartitioner

**Importance:** medium

Partitioner class that implements the **org.apache.kafka.clients.producer.Partitioner** interface.

### **receive.buffer.bytes**

**Type:** int

**Default:** 32768

**Valid Values:** [-1,...]

**Importance:** medium

The size of the TCP receive buffer (SO\_RCVBUF) to use when reading data. If the value is -1, the OS default will be used.

### **request.timeout.ms**

**Type:** int

**Default:** 30000

**Valid Values:** [0,...]

**Importance:** medium

The configuration controls the maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted. This should be larger than **replica.lag.time.max.ms** (a broker configuration) to reduce the possibility of message duplication due to unnecessary producer retries.

### **sasl.client.callback.handler.class**

**Type:** class

**Default:** null

**Importance:** medium

The fully qualified name of a SASL client callback handler class that implements the AuthenticateCallbackHandler interface.

### **sasl.jaas.config**

**Type:** password

**Default:** null

**Importance:** medium

JAAS login context parameters for SASL connections in the format used by JAAS configuration files. JAAS configuration file format is described [here](#). The format for the value is: 'loginModuleClass controlFlag (optionName=optionValue)\*;'. For brokers, the config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, listener.name.sasl\_ssl.scram-sha-256.sasl.jaas.config=com.example.ScramLoginModule required;

### **sasl.kerberos.service.name**

**Type:** string

**Default:** null

**Importance:** medium

The Kerberos principal name that Kafka runs as. This can be defined either in Kafka's JAAS config or in Kafka's config.

### **sasl.login.callback.handler.class**

**Type:** class

**Default:** null

**Importance:** medium

The fully qualified name of a SASL login callback handler class that implements the `AuthenticateCallbackHandler` interface. For brokers, login callback handler config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.login.callback.handler.class=com.example.CustomScramLoginCallbackHandler`.

### **sasl.login.class**

**Type:** class

**Default:** null

**Importance:** medium

The fully qualified name of a class that implements the `Login` interface. For brokers, login config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.login.class=com.example.CustomScramLogin`.

### **sasl.mechanism**

**Type:** string

**Default:** GSSAPI

**Importance:** medium

SASL mechanism used for client connections. This may be any mechanism for which a security provider is available. GSSAPI is the default mechanism.

### **security.protocol**

**Type:** string

**Default:** PLAINTEXT

**Importance:** medium

Protocol used to communicate with brokers. Valid values are: `PLAINTEXT`, `SSL`, `SASL_PLAINTEXT`, `SASL_SSL`.

### **send.buffer.bytes**

**Type:** int

**Default:** 131072

**Valid Values:** [-1,...]

**Importance:** medium

The size of the TCP send buffer (`SO_SNDBUF`) to use when sending data. If the value is -1, the OS default will be used.

### **ssl.enabled.protocols**

**Type:** list

**Default:** TLSv1.2,TLSv1.1,TLSv1

**Importance:** medium

The list of protocols enabled for SSL connections.

**ssl.keystore.type**

**Type:** string

**Default:** JKS

**Importance:** medium

The file format of the key store file. This is optional for client.

**ssl.protocol**

**Type:** string

**Default:** TLS

**Importance:** medium

The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for most cases. Allowed values in recent JVMs are TLS, TLSv1.1 and TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to known security vulnerabilities.

**ssl.provider**

**Type:** string

**Default:** null

**Importance:** medium

The name of the security provider used for SSL connections. Default value is the default security provider of the JVM.

**ssl.truststore.type**

**Type:** string

**Default:** JKS

**Importance:** medium

The file format of the trust store file.

**enable.idempotence**

**Type:** boolean

**Default:** false

**Importance:** low

When set to 'true', the producer will ensure that exactly one copy of each message is written in the stream. If 'false', producer retries due to broker failures, etc., may write duplicates of the retried message in the stream. Note that enabling idempotence requires **max.in.flight.requests.per.connection** to be less than or equal to 5, **retries** to be greater than 0 and **acks** must be 'all'. If these values are not explicitly set by the user, suitable values will be chosen. If incompatible values are set, a **ConfigException** will be thrown.

**interceptor.classes**

**Type:** list

**Default:** ""

**Valid Values:** non-null string

**Importance:** low

A list of classes to use as interceptors. Implementing the **org.apache.kafka.clients.producer.ProducerInterceptor** interface allows you to intercept (and possibly mutate) the records received by the producer before they are published to the Kafka cluster. By default, there are no interceptors.

**max.in.flight.requests.per.connection**

**Type:** int

**Default:** 5

**Valid Values:** [1,...]

**Importance:** low

The maximum number of unacknowledged requests the client will send on a single connection before blocking. Note that if this setting 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., if retries are enabled).

### **metadata.max.age.ms**

**Type:** long

**Default:** 300000

**Valid Values:** [0,...]

**Importance:** low

The period of time in milliseconds after which we force a refresh of metadata even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions.

### **metric.reporters**

**Type:** list

**Default:** ""

**Valid Values:** non-null string

**Importance:** low

A list of classes to use as metrics reporters. Implementing the **org.apache.kafka.common.metrics.MetricsReporter** interface allows plugging in classes that will be notified of new metric creation. The JmxReporter is always included to register JMX statistics.

### **metrics.num.samples**

**Type:** int

**Default:** 2

**Valid Values:** [1,...]

**Importance:** low

The number of samples maintained to compute metrics.

### **metrics.recording.level**

**Type:** string

**Default:** INFO

**Valid Values:** [INFO, DEBUG]

**Importance:** low

The highest recording level for metrics.

### **metrics.sample.window.ms**

**Type:** long

**Default:** 30000

**Valid Values:** [0,...]

**Importance:** low

The window of time a metrics sample is computed over.

### **reconnect.backoff.max.ms**

**Type:** long

**Default:** 1000

**Valid Values:** [0,...]

**Importance:** low

The maximum amount of time in milliseconds to wait when reconnecting to a broker that has repeatedly failed to connect. If provided, the backoff per host will increase exponentially for each consecutive connection failure, up to this maximum. After calculating the backoff increase, 20%

random jitter is added to avoid connection storms.

### **reconnect.backoff.ms**

**Type:** long

**Default:** 50

**Valid Values:** [0,...]

**Importance:** low

The base amount of time to wait before attempting to reconnect to a given host. This avoids repeatedly connecting to a host in a tight loop. This backoff applies to all connection attempts by the client to a broker.

### **retry.backoff.ms**

**Type:** long

**Default:** 100

**Valid Values:** [0,...]

**Importance:** low

The amount of time to wait before attempting to retry a failed request to a given topic partition. This avoids repeatedly sending requests in a tight loop under some failure scenarios.

### **sasl.kerberos.kinit.cmd**

**Type:** string

**Default:** /usr/bin/kinit

**Importance:** low

Kerberos kinit command path.

### **sasl.kerberos.min.time.before.relogin**

**Type:** long

**Default:** 60000

**Importance:** low

Login thread sleep time between refresh attempts.

### **sasl.kerberos.ticket.renew.jitter**

**Type:** double

**Default:** 0.05

**Importance:** low

Percentage of random jitter added to the renewal time.

### **sasl.kerberos.ticket.renew.window.factor**

**Type:** double

**Default:** 0.8

**Importance:** low

Login thread will sleep until the specified window factor of time from last refresh to ticket's expiry has been reached, at which time it will try to renew the ticket.

### **sasl.login.refresh.buffer.seconds**

**Type:** short

**Default:** 300

**Valid Values:** [0,...,3600]

**Importance:** low

The amount of buffer time before credential expiration to maintain when refreshing a credential, in seconds. If a refresh would otherwise occur closer to expiration than the number of buffer seconds

then the refresh will be moved up to maintain as much of the buffer time as possible. Legal values are between 0 and 3600 (1 hour); a default value of 300 (5 minutes) is used if no value is specified. This value and `sasl.login.refresh.min.period.seconds` are both ignored if their sum exceeds the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

### **sasl.login.refresh.min.period.seconds**

**Type:** short

**Default:** 60

**Valid Values:** [0,...,900]

**Importance:** low

The desired minimum time for the login refresh thread to wait before refreshing a credential, in seconds. Legal values are between 0 and 900 (15 minutes); a default value of 60 (1 minute) is used if no value is specified. This value and `sasl.login.refresh.buffer.seconds` are both ignored if their sum exceeds the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

### **sasl.login.refresh.window.factor**

**Type:** double

**Default:** 0.8

**Valid Values:** [0.5,...,1.0]

**Importance:** low

Login refresh thread will sleep until the specified window factor relative to the credential's lifetime has been reached, at which time it will try to refresh the credential. Legal values are between 0.5 (50%) and 1.0 (100%) inclusive; a default value of 0.8 (80%) is used if no value is specified. Currently applies only to OAUTHBEARER.

### **sasl.login.refresh.window.jitter**

**Type:** double

**Default:** 0.05

**Valid Values:** [0.0,...,0.25]

**Importance:** low

The maximum amount of random jitter relative to the credential's lifetime that is added to the login refresh thread's sleep time. Legal values are between 0 and 0.25 (25%) inclusive; a default value of 0.05 (5%) is used if no value is specified. Currently applies only to OAUTHBEARER.

### **ssl.cipher.suites**

**Type:** list

**Default:** null

**Importance:** low

A list of cipher suites. This is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol. By default all the available cipher suites are supported.

### **ssl.endpoint.identification.algorithm**

**Type:** string

**Default:** https

**Importance:** low

The endpoint identification algorithm to validate server hostname using server certificate.

### **ssl.keymanager.algorithm**

**Type:** string

**Default:** SunX509

**Importance:** low

The algorithm used by key manager factory for SSL connections. Default value is the key manager factory algorithm configured for the Java Virtual Machine.

### **ssl.secure.random.implementation**

**Type:** string

**Default:** null

**Importance:** low

The SecureRandom PRNG implementation to use for SSL cryptography operations.

### **ssl.trustmanager.algorithm**

**Type:** string

**Default:** PKIX

**Importance:** low

The algorithm used by trust manager factory for SSL connections. Default value is the trust manager factory algorithm configured for the Java Virtual Machine.

### **transaction.timeout.ms**

**Type:** int

**Default:** 60000

**Importance:** low

The maximum amount of time in ms that the transaction coordinator will wait for a transaction status update from the producer before proactively aborting the ongoing transaction. If this value is larger than the `transaction.max.timeout.ms` setting in the broker, the request will fail with a

**InvalidTransactionTimeout** error.

### **transactional.id**

**Type:** string

**Default:** null

**Valid Values:** non-empty string

**Importance:** low

The TransactionalId to use for transactional delivery. This enables reliability semantics which span multiple producer sessions since it allows the client to guarantee that transactions using the same TransactionalId have been completed prior to starting any new transactions. If no TransactionalId is provided, then the producer is limited to idempotent delivery. Note that **enable.idempotence** must be enabled if a TransactionalId is configured. The default is **null**, which means transactions cannot be used. Note that, by default, transactions require a cluster of at least three brokers which is the recommended setting for production; for development you can change this, by adjusting broker setting **transaction.state.log.replication.factor**.

## APPENDIX E. ADMIN CLIENT CONFIGURATION PARAMETERS

### **bootstrap.servers**

**Type:** list

**Importance:** high

A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. The client will make use of all servers irrespective of which servers are specified here for bootstrapping—this list only impacts the initial hosts used to discover the full set of servers. This list should be in the form **host1:port1,host2:port2,...** Since these servers are just used for the initial connection to discover the full cluster membership (which may change dynamically), this list need not contain the full set of servers (you may want more than one, though, in case a server is down).

### **ssl.key.password**

**Type:** password

**Default:** null

**Importance:** high

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

### **ssl.keystore.location**

**Type:** string

**Default:** null

**Importance:** high

The location of the key store file. This is optional for client and can be used for two-way authentication for client.

### **ssl.keystore.password**

**Type:** password

**Default:** null

**Importance:** high

The store password for the key store file. This is optional for client and only needed if `ssl.keystore.location` is configured.

### **ssl.truststore.location**

**Type:** string

**Default:** null

**Importance:** high

The location of the trust store file.

### **ssl.truststore.password**

**Type:** password

**Default:** null

**Importance:** high

The password for the trust store file. If a password is not set access to the truststore is still available, but integrity checking is disabled.

### **client.dns.lookup**

**Type:** string

**Default:** default

**Valid Values:** [default, use\_all\_dns\_ips, resolve\_canonical\_bootstrap\_servers\_only]

**Importance:** medium

Controls how the client uses DNS lookups.



If set to **use\_all\_dns\_ips** then, when the lookup returns multiple IP addresses for a hostname, they will all be attempted to connect to before failing the connection. Applies to both bootstrap and advertised servers.

If the value is **resolve\_canonical\_bootstrap\_servers\_only** each entry will be resolved and expanded into a list of canonical names.

### **client.id**

**Type:** string

**Default:** ""

**Importance:** medium

An id string to pass to the server when making requests. The purpose of this is to be able to track the source of requests beyond just ip/port by allowing a logical application name to be included in server-side request logging.

### **connections.max.idle.ms**

**Type:** long

**Default:** 300000

**Importance:** medium

Close idle connections after the number of milliseconds specified by this config.

### **receive.buffer.bytes**

**Type:** int

**Default:** 65536

**Valid Values:** [-1,...]

**Importance:** medium

The size of the TCP receive buffer (SO\_RCVBUF) to use when reading data. If the value is -1, the OS default will be used.

### **request.timeout.ms**

**Type:** int

**Default:** 120000

**Valid Values:** [0,...]

**Importance:** medium

The configuration controls the maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.

### **sasl.client.callback.handler.class**

**Type:** class

**Default:** null

**Importance:** medium

The fully qualified name of a SASL client callback handler class that implements the AuthenticateCallbackHandler interface.

### **sasl.jaas.config**

**Type:** password

**Default:** null

**Importance:** medium

JAAS login context parameters for SASL connections in the format used by JAAS configuration files. JAAS configuration file format is described [here](#). The format for the value is: 'loginModuleClass controlFlag (optionName=optionValue)\*;'. For brokers, the config must be prefixed with listener

prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.jaas.config=com.example.ScramLoginModule` required;

### **sasl.kerberos.service.name**

**Type:** string

**Default:** null

**Importance:** medium

The Kerberos principal name that Kafka runs as. This can be defined either in Kafka's JAAS config or in Kafka's config.

### **sasl.login.callback.handler.class**

**Type:** class

**Default:** null

**Importance:** medium

The fully qualified name of a SASL login callback handler class that implements the `AuthenticateCallbackHandler` interface. For brokers, login callback handler config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.login.callback.handler.class=com.example.CustomScramLoginCallbackHandler`.

### **sasl.login.class**

**Type:** class

**Default:** null

**Importance:** medium

The fully qualified name of a class that implements the `Login` interface. For brokers, login config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.login.class=com.example.CustomScramLogin`.

### **sasl.mechanism**

**Type:** string

**Default:** GSSAPI

**Importance:** medium

SASL mechanism used for client connections. This may be any mechanism for which a security provider is available. GSSAPI is the default mechanism.

### **security.protocol**

**Type:** string

**Default:** PLAINTEXT

**Importance:** medium

Protocol used to communicate with brokers. Valid values are: `PLAINTEXT`, `SSL`, `SASL_PLAINTEXT`, `SASL_SSL`.

### **send.buffer.bytes**

**Type:** int

**Default:** 131072

**Valid Values:** [-1,...]

**Importance:** medium

The size of the TCP send buffer (`SO_SNDBUF`) to use when sending data. If the value is -1, the OS default will be used.

### **ssl.enabled.protocols**

**Type:** list

**Default:** TLSv1.2,TLSv1.1,TLSv1

**Importance:** medium

The list of protocols enabled for SSL connections.

### **ssl.keystore.type**

**Type:** string

**Default:** JKS

**Importance:** medium

The file format of the key store file. This is optional for client.

### **ssl.protocol**

**Type:** string

**Default:** TLS

**Importance:** medium

The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for most cases. Allowed values in recent JVMs are TLS, TLSv1.1 and TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to known security vulnerabilities.

### **ssl.provider**

**Type:** string

**Default:** null

**Importance:** medium

The name of the security provider used for SSL connections. Default value is the default security provider of the JVM.

### **ssl.truststore.type**

**Type:** string

**Default:** JKS

**Importance:** medium

The file format of the trust store file.

### **metadata.max.age.ms**

**Type:** long

**Default:** 300000

**Valid Values:** [0,...]

**Importance:** low

The period of time in milliseconds after which we force a refresh of metadata even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions.

### **metric.reporters**

**Type:** list

**Default:** ""

**Importance:** low

A list of classes to use as metrics reporters. Implementing the **org.apache.kafka.common.metrics.MetricsReporter** interface allows plugging in classes that will be notified of new metric creation. The JmxReporter is always included to register JMX statistics.

### **metrics.num.samples**

**Type:** int

**Default:** 2

**Valid Values:** [1,...]

**Importance:** low

The number of samples maintained to compute metrics.

### **metrics.recording.level**

**Type:** string

**Default:** INFO

**Valid Values:** [INFO, DEBUG]

**Importance:** low

The highest recording level for metrics.

### **metrics.sample.window.ms**

**Type:** long

**Default:** 30000

**Valid Values:** [0,...]

**Importance:** low

The window of time a metrics sample is computed over.

### **reconnect.backoff.max.ms**

**Type:** long

**Default:** 1000

**Valid Values:** [0,...]

**Importance:** low

The maximum amount of time in milliseconds to wait when reconnecting to a broker that has repeatedly failed to connect. If provided, the backoff per host will increase exponentially for each consecutive connection failure, up to this maximum. After calculating the backoff increase, 20% random jitter is added to avoid connection storms.

### **reconnect.backoff.ms**

**Type:** long

**Default:** 50

**Valid Values:** [0,...]

**Importance:** low

The base amount of time to wait before attempting to reconnect to a given host. This avoids repeatedly connecting to a host in a tight loop. This backoff applies to all connection attempts by the client to a broker.

### **retries**

**Type:** int

**Default:** 5

**Valid Values:** [0,...]

**Importance:** low

Setting a value greater than zero will cause the client to resend any request that fails with a potentially transient error.

### **retry.backoff.ms**

**Type:** long

**Default:** 100

**Valid Values:** [0,...]

**Importance:** low

The amount of time to wait before attempting to retry a failed request. This avoids repeatedly sending requests in a tight loop under some failure scenarios.

**sasl.kerberos.kinit.cmd**

**Type:** string

**Default:** /usr/bin/kinit

**Importance:** low

Kerberos kinit command path.

**sasl.kerberos.min.time.before.relogin**

**Type:** long

**Default:** 60000

**Importance:** low

Login thread sleep time between refresh attempts.

**sasl.kerberos.ticket.renew.jitter**

**Type:** double

**Default:** 0.05

**Importance:** low

Percentage of random jitter added to the renewal time.

**sasl.kerberos.ticket.renew.window.factor**

**Type:** double

**Default:** 0.8

**Importance:** low

Login thread will sleep until the specified window factor of time from last refresh to ticket's expiry has been reached, at which time it will try to renew the ticket.

**sasl.login.refresh.buffer.seconds**

**Type:** short

**Default:** 300

**Valid Values:** [0,...,3600]

**Importance:** low

The amount of buffer time before credential expiration to maintain when refreshing a credential, in seconds. If a refresh would otherwise occur closer to expiration than the number of buffer seconds then the refresh will be moved up to maintain as much of the buffer time as possible. Legal values are between 0 and 3600 (1 hour); a default value of 300 (5 minutes) is used if no value is specified. This value and `sasl.login.refresh.min.period.seconds` are both ignored if their sum exceeds the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

**sasl.login.refresh.min.period.seconds**

**Type:** short

**Default:** 60

**Valid Values:** [0,...,900]

**Importance:** low

The desired minimum time for the login refresh thread to wait before refreshing a credential, in seconds. Legal values are between 0 and 900 (15 minutes); a default value of 60 (1 minute) is used if no value is specified. This value and `sasl.login.refresh.buffer.seconds` are both ignored if their sum exceeds the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

**sasl.login.refresh.window.factor**

**Type:** double

**Default:** 0.8

**Valid Values:** [0.5,...,1.0]

**Importance:** low

Login refresh thread will sleep until the specified window factor relative to the credential's lifetime has been reached, at which time it will try to refresh the credential. Legal values are between 0.5 (50%) and 1.0 (100%) inclusive; a default value of 0.8 (80%) is used if no value is specified. Currently applies only to OAUTHBEARER.

**sasl.login.refresh.window.jitter****Type:** double**Default:** 0.05**Valid Values:** [0.0,...,0.25]**Importance:** low

The maximum amount of random jitter relative to the credential's lifetime that is added to the login refresh thread's sleep time. Legal values are between 0 and 0.25 (25%) inclusive; a default value of 0.05 (5%) is used if no value is specified. Currently applies only to OAUTHBEARER.

**ssl.cipher.suites****Type:** list**Default:** null**Importance:** low

A list of cipher suites. This is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol. By default all the available cipher suites are supported.

**ssl.endpoint.identification.algorithm****Type:** string**Default:** https**Importance:** low

The endpoint identification algorithm to validate server hostname using server certificate.

**ssl.keymanager.algorithm****Type:** string**Default:** SunX509**Importance:** low

The algorithm used by key manager factory for SSL connections. Default value is the key manager factory algorithm configured for the Java Virtual Machine.

**ssl.secure.random.implementation****Type:** string**Default:** null**Importance:** low

The SecureRandom PRNG implementation to use for SSL cryptography operations.

**ssl.trustmanager.algorithm****Type:** string**Default:** PKIX**Importance:** low

The algorithm used by trust manager factory for SSL connections. Default value is the trust manager factory algorithm configured for the Java Virtual Machine.

## APPENDIX F. KAFKA CONNECT CONFIGURATION PARAMETERS

### **config.storage.topic**

**Type:** string

**Importance:** high

The name of the Kafka topic where connector configurations are stored.

### **group.id**

**Type:** string

**Importance:** high

A unique string that identifies the Connect cluster group this worker belongs to.

### **key.converter**

**Type:** class

**Importance:** high

Converter class used to convert between Kafka Connect format and the serialized form that is written to Kafka. This controls the format of the keys in messages written to or read from Kafka, and since this is independent of connectors it allows any connector to work with any serialization format. Examples of common formats include JSON and Avro.

### **offset.storage.topic**

**Type:** string

**Importance:** high

The name of the Kafka topic where connector offsets are stored.

### **status.storage.topic**

**Type:** string

**Importance:** high

The name of the Kafka topic where connector and task status are stored.

### **value.converter**

**Type:** class

**Importance:** high

Converter class used to convert between Kafka Connect format and the serialized form that is written to Kafka. This controls the format of the values in messages written to or read from Kafka, and since this is independent of connectors it allows any connector to work with any serialization format. Examples of common formats include JSON and Avro.

### **bootstrap.servers**

**Type:** list

**Default:** localhost:9092

**Importance:** high

A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. The client will make use of all servers irrespective of which servers are specified here for bootstrapping—this list only impacts the initial hosts used to discover the full set of servers. This list should be in the form **host1:port1,host2:port2,...** Since these servers are just used for the initial connection to discover the full cluster membership (which may change dynamically), this list need not contain the full set of servers (you may want more than one, though, in case a server is down).

**heartbeat.interval.ms****Type:** int**Default:** 3000**Importance:** high

The expected time between heartbeats to the group coordinator when using Kafka's group management facilities. Heartbeats are used to ensure that the worker's session stays active and to facilitate rebalancing when new members join or leave the group. The value must be set lower than **session.timeout.ms**, but typically should be set no higher than 1/3 of that value. It can be adjusted even lower to control the expected time for normal rebalances.

**rebalance.timeout.ms****Type:** int**Default:** 60000**Importance:** high

The maximum allowed time for each worker to join the group once a rebalance has begun. This is basically a limit on the amount of time needed for all tasks to flush any pending data and commit offsets. If the timeout is exceeded, then the worker will be removed from the group, which will cause offset commit failures.

**session.timeout.ms****Type:** int**Default:** 10000**Importance:** high

The timeout used to detect worker failures. The worker sends periodic heartbeats to indicate its liveness to the broker. If no heartbeats are received by the broker before the expiration of this session timeout, then the broker will remove the worker from the group and initiate a rebalance. Note that the value must be in the allowable range as configured in the broker configuration by **group.min.session.timeout.ms** and **group.max.session.timeout.ms**.

**ssl.key.password****Type:** password**Default:** null**Importance:** high

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

**ssl.keystore.location****Type:** string**Default:** null**Importance:** high

The location of the key store file. This is optional for client and can be used for two-way authentication for client.

**ssl.keystore.password****Type:** password**Default:** null**Importance:** high

The store password for the key store file. This is optional for client and only needed if **ssl.keystore.location** is configured.

**ssl.truststore.location**



**Type:** string

**Default:** null

**Importance:** high

The location of the trust store file.

### **ssl.truststore.password**

**Type:** password

**Default:** null

**Importance:** high

The password for the trust store file. If a password is not set access to the truststore is still available, but integrity checking is disabled.

### **client.dns.lookup**

**Type:** string

**Default:** default

**Valid Values:** [default, use\_all\_dns\_ips, resolve\_canonical\_bootstrap\_servers\_only]

**Importance:** medium

Controls how the client uses DNS lookups.

If set to **use\_all\_dns\_ips** then, when the lookup returns multiple IP addresses for a hostname, they will all be attempted to connect to before failing the connection. Applies to both bootstrap and advertised servers.

If the value is **resolve\_canonical\_bootstrap\_servers\_only** each entry will be resolved and expanded into a list of canonical names.

### **connections.max.idle.ms**

**Type:** long

**Default:** 540000

**Importance:** medium

Close idle connections after the number of milliseconds specified by this config.

### **connector.client.config.override.policy**

**Type:** string

**Default:** None

**Importance:** medium

Class name or alias of implementation of **ConnectorClientConfigOverridePolicy**. Defines what client configurations can be overridden by the connector. The default implementation is **None**. The other possible policies in the framework include **All** and **Principal**.

### **receive.buffer.bytes**

**Type:** int

**Default:** 32768

**Valid Values:** [0,...]

**Importance:** medium

The size of the TCP receive buffer (SO\_RCVBUF) to use when reading data. If the value is -1, the OS default will be used.

### **request.timeout.ms**

**Type:** int

**Default:** 40000

**Valid Values:** [0,...]

**Importance:** medium

The configuration controls the maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.

### **sasl.client.callback.handler.class**

**Type:** class

**Default:** null

**Importance:** medium

The fully qualified name of a SASL client callback handler class that implements the `AuthenticateCallbackHandler` interface.

### **sasl.jaas.config**

**Type:** password

**Default:** null

**Importance:** medium

JAAS login context parameters for SASL connections in the format used by JAAS configuration files. JAAS configuration file format is described [here](#). The format for the value is: 'loginModuleClass controlFlag (optionName=optionValue)\*;'. For brokers, the config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.jaas.config=com.example.ScramLoginModule required;`

### **sasl.kerberos.service.name**

**Type:** string

**Default:** null

**Importance:** medium

The Kerberos principal name that Kafka runs as. This can be defined either in Kafka's JAAS config or in Kafka's config.

### **sasl.login.callback.handler.class**

**Type:** class

**Default:** null

**Importance:** medium

The fully qualified name of a SASL login callback handler class that implements the `AuthenticateCallbackHandler` interface. For brokers, login callback handler config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.login.callback.handler.class=com.example.CustomScramLoginCallbackHandler.`

### **sasl.login.class**

**Type:** class

**Default:** null

**Importance:** medium

The fully qualified name of a class that implements the `Login` interface. For brokers, login config must be prefixed with listener prefix and SASL mechanism name in lower-case. For example, `listener.name.sasl_ssl.scram-sha-256.sasl.login.class=com.example.CustomScramLogin.`

### **sasl.mechanism**

**Type:** string

**Default:** GSSAPI

**Importance:** medium

SASL mechanism used for client connections. This may be any mechanism for which a security provider is available. GSSAPI is the default mechanism.

### **security.protocol**

**Type:** string

**Default:** PLAINTEXT

**Importance:** medium

Protocol used to communicate with brokers. Valid values are: PLAINTEXT, SSL, SASL\_PLAINTEXT, SASL\_SSL.

### **send.buffer.bytes**

**Type:** int

**Default:** 131072

**Valid Values:** [0,...]

**Importance:** medium

The size of the TCP send buffer (SO\_SNDBUF) to use when sending data. If the value is -1, the OS default will be used.

### **ssl.enabled.protocols**

**Type:** list

**Default:** TLSv1.2,TLSv1.1,TLSv1

**Importance:** medium

The list of protocols enabled for SSL connections.

### **ssl.keystore.type**

**Type:** string

**Default:** JKS

**Importance:** medium

The file format of the key store file. This is optional for client.

### **ssl.protocol**

**Type:** string

**Default:** TLS

**Importance:** medium

The SSL protocol used to generate the SSLContext. Default setting is TLS, which is fine for most cases. Allowed values in recent JVMs are TLS, TLSv1.1 and TLSv1.2. SSL, SSLv2 and SSLv3 may be supported in older JVMs, but their usage is discouraged due to known security vulnerabilities.

### **ssl.provider**

**Type:** string

**Default:** null

**Importance:** medium

The name of the security provider used for SSL connections. Default value is the default security provider of the JVM.

### **ssl.truststore.type**

**Type:** string

**Default:** JKS

**Importance:** medium

The file format of the trust store file.

**worker.sync.timeout.ms****Type:** int**Default:** 3000**Importance:** medium

When the worker is out of sync with other workers and needs to resynchronize configurations, wait up to this amount of time before giving up, leaving the group, and waiting a backoff period before rejoining.

**worker.unsync.backoff.ms****Type:** int**Default:** 300000**Importance:** medium

When the worker is out of sync with other workers and fails to catch up within `worker.sync.timeout.ms`, leave the Connect cluster for this long before rejoining.

**access.control.allow.methods****Type:** string**Default:** ""**Importance:** low

Sets the methods supported for cross origin requests by setting the `Access-Control-Allow-Methods` header. The default value of the `Access-Control-Allow-Methods` header allows cross origin requests for GET, POST and HEAD.

**access.control.allow.origin****Type:** string**Default:** ""**Importance:** low

Value to set the `Access-Control-Allow-Origin` header to for REST API requests. To enable cross origin access, set this to the domain of the application that should be permitted to access the API, or '\*' to allow access from any domain. The default value only allows access from the domain of the REST API.

**client.id****Type:** string**Default:** ""**Importance:** low

An id string to pass to the server when making requests. The purpose of this is to be able to track the source of requests beyond just ip/port by allowing a logical application name to be included in server-side request logging.

**config.providers****Type:** list**Default:** ""**Importance:** low

Comma-separated names of **ConfigProvider** classes, loaded and used in the order specified. Implementing the interface **ConfigProvider** allows you to replace variable references in connector configurations, such as for externalized secrets.

**config.storage.replication.factor****Type:** short**Default:** 3

**Valid Values:** [1,...]

**Importance:** low

Replication factor used when creating the configuration storage topic.

### **connect.protocol**

**Type:** string

**Default:** compatible

**Valid Values:** [eager, compatible]

**Importance:** low

Compatibility mode for Kafka Connect Protocol.

### **header.converter**

**Type:** class

**Default:** org.apache.kafka.connect.storage.SimpleHeaderConverter

**Importance:** low

HeaderConverter class used to convert between Kafka Connect format and the serialized form that is written to Kafka. This controls the format of the header values in messages written to or read from Kafka, and since this is independent of connectors it allows any connector to work with any serialization format. Examples of common formats include JSON and Avro. By default, the SimpleHeaderConverter is used to serialize header values to strings and deserialize them by inferring the schemas.

### **internal.key.converter**

**Type:** class

**Default:** org.apache.kafka.connect.json.JsonConverter

**Importance:** low

Converter class used to convert between Kafka Connect format and the serialized form that is written to Kafka. This controls the format of the keys in messages written to or read from Kafka, and since this is independent of connectors it allows any connector to work with any serialization format. Examples of common formats include JSON and Avro. This setting controls the format used for internal bookkeeping data used by the framework, such as configs and offsets, so users can typically use any functioning Converter implementation. Deprecated; will be removed in an upcoming version.

### **internal.value.converter**

**Type:** class

**Default:** org.apache.kafka.connect.json.JsonConverter

**Importance:** low

Converter class used to convert between Kafka Connect format and the serialized form that is written to Kafka. This controls the format of the values in messages written to or read from Kafka, and since this is independent of connectors it allows any connector to work with any serialization format. Examples of common formats include JSON and Avro. This setting controls the format used for internal bookkeeping data used by the framework, such as configs and offsets, so users can typically use any functioning Converter implementation. Deprecated; will be removed in an upcoming version.

### **listeners**

**Type:** list

**Default:** null

**Importance:** low

List of comma-separated URIs the REST API will listen on. The supported protocols are HTTP and HTTPS. Specify hostname as 0.0.0.0 to bind to all interfaces. Leave hostname empty to bind to default interface. Examples of legal listener lists: HTTP://myhost:8083,HTTPS://myhost:8084.

**metadata.max.age.ms****Type:** long**Default:** 300000**Valid Values:** [0,...]**Importance:** low

The period of time in milliseconds after which we force a refresh of metadata even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions.

**metric.reporters****Type:** list**Default:** ""**Importance:** low

A list of classes to use as metrics reporters. Implementing the **org.apache.kafka.common.metrics.MetricsReporter** interface allows plugging in classes that will be notified of new metric creation. The JmxReporter is always included to register JMX statistics.

**metrics.num.samples****Type:** int**Default:** 2**Valid Values:** [1,...]**Importance:** low

The number of samples maintained to compute metrics.

**metrics.recording.level****Type:** string**Default:** INFO**Valid Values:** [INFO, DEBUG]**Importance:** low

The highest recording level for metrics.

**metrics.sample.window.ms****Type:** long**Default:** 30000**Valid Values:** [0,...]**Importance:** low

The window of time a metrics sample is computed over.

**offset.flush.interval.ms****Type:** long**Default:** 60000**Importance:** low

Interval at which to try committing offsets for tasks.

**offset.flush.timeout.ms****Type:** long**Default:** 5000**Importance:** low

Maximum number of milliseconds to wait for records to flush and partition offset data to be committed to offset storage before cancelling the process and restoring the offset data to be committed in a future attempt.

**offset.storage.partitions****Type:** int**Default:** 25**Valid Values:** [1,...]**Importance:** low

The number of partitions used when creating the offset storage topic.

**offset.storage.replication.factor****Type:** short**Default:** 3**Valid Values:** [1,...]**Importance:** low

Replication factor used when creating the offset storage topic.

**plugin.path****Type:** list**Default:** null**Importance:** low

List of paths separated by commas (,) that contain plugins (connectors, converters, transformations). The list should consist of top level directories that include any combination of: a) directories immediately containing jars with plugins and their dependencies b) uber-jars with plugins and their dependencies c) directories immediately containing the package directory structure of classes of plugins and their dependencies Note: symlinks will be followed to discover dependencies or plugins. Examples:

plugin.path=/usr/local/share/java,/usr/local/share/kafka/plugins,/opt/connectors.

**reconnect.backoff.max.ms****Type:** long**Default:** 1000**Valid Values:** [0,...]**Importance:** low

The maximum amount of time in milliseconds to wait when reconnecting to a broker that has repeatedly failed to connect. If provided, the backoff per host will increase exponentially for each consecutive connection failure, up to this maximum. After calculating the backoff increase, 20% random jitter is added to avoid connection storms.

**reconnect.backoff.ms****Type:** long**Default:** 50**Valid Values:** [0,...]**Importance:** low

The base amount of time to wait before attempting to reconnect to a given host. This avoids repeatedly connecting to a host in a tight loop. This backoff applies to all connection attempts by the client to a broker.

**rest.advertised.host.name****Type:** string**Default:** null**Importance:** low

If this is set, this is the hostname that will be given out to other workers to connect to.

**rest.advertised.listener**

**Type:** string

**Default:** null

**Importance:** low

Sets the advertised listener (HTTP or HTTPS) which will be given to other workers to use.

### **rest.advertised.port**

**Type:** int

**Default:** null

**Importance:** low

If this is set, this is the port that will be given out to other workers to connect to.

### **rest.extension.classes**

**Type:** list

**Default:** ""

**Importance:** low

Comma-separated names of **ConnectRestExtension** classes, loaded and called in the order specified. Implementing the interface **ConnectRestExtension** allows you to inject into Connect's REST API user defined resources like filters. Typically used to add custom capability like logging, security, etc.

### **rest.host.name**

**Type:** string

**Default:** null

**Importance:** low

Hostname for the REST API. If this is set, it will only bind to this interface.

### **rest.port**

**Type:** int

**Default:** 8083

**Importance:** low

Port for the REST API to listen on.

### **retry.backoff.ms**

**Type:** long

**Default:** 100

**Valid Values:** [0,...]

**Importance:** low

The amount of time to wait before attempting to retry a failed request to a given topic partition. This avoids repeatedly sending requests in a tight loop under some failure scenarios.

### **sasl.kerberos.kinit.cmd**

**Type:** string

**Default:** /usr/bin/kinit

**Importance:** low

Kerberos kinit command path.

### **sasl.kerberos.min.time.before.relogin**

**Type:** long

**Default:** 60000

**Importance:** low

Login thread sleep time between refresh attempts.



**sasl.kerberos.ticket.renew.jitter**

**Type:** double

**Default:** 0.05

**Importance:** low

Percentage of random jitter added to the renewal time.

**sasl.kerberos.ticket.renew.window.factor**

**Type:** double

**Default:** 0.8

**Importance:** low

Login thread will sleep until the specified window factor of time from last refresh to ticket's expiry has been reached, at which time it will try to renew the ticket.

**sasl.login.refresh.buffer.seconds**

**Type:** short

**Default:** 300

**Valid Values:** [0,...,3600]

**Importance:** low

The amount of buffer time before credential expiration to maintain when refreshing a credential, in seconds. If a refresh would otherwise occur closer to expiration than the number of buffer seconds then the refresh will be moved up to maintain as much of the buffer time as possible. Legal values are between 0 and 3600 (1 hour); a default value of 300 (5 minutes) is used if no value is specified. This value and `sasl.login.refresh.min.period.seconds` are both ignored if their sum exceeds the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

**sasl.login.refresh.min.period.seconds**

**Type:** short

**Default:** 60

**Valid Values:** [0,...,900]

**Importance:** low

The desired minimum time for the login refresh thread to wait before refreshing a credential, in seconds. Legal values are between 0 and 900 (15 minutes); a default value of 60 (1 minute) is used if no value is specified. This value and `sasl.login.refresh.buffer.seconds` are both ignored if their sum exceeds the remaining lifetime of a credential. Currently applies only to OAUTHBEARER.

**sasl.login.refresh.window.factor**

**Type:** double

**Default:** 0.8

**Valid Values:** [0.5,...,1.0]

**Importance:** low

Login refresh thread will sleep until the specified window factor relative to the credential's lifetime has been reached, at which time it will try to refresh the credential. Legal values are between 0.5 (50%) and 1.0 (100%) inclusive; a default value of 0.8 (80%) is used if no value is specified. Currently applies only to OAUTHBEARER.

**sasl.login.refresh.window.jitter**

**Type:** double

**Default:** 0.05

**Valid Values:** [0.0,...,0.25]

**Importance:** low

The maximum amount of random jitter relative to the credential's lifetime that is added to the login refresh thread's sleep time. Legal values are between 0 and 0.25 (25%) inclusive; a default value of 0.05 (5%) is used if no value is specified. Currently applies only to OAUTHBEARER.

### **scheduled.rebalance.max.delay.ms**

**Type:** int

**Default:** 300000

**Valid Values:** [0,...,2147483647]

**Importance:** low

Compatibility mode for Kafka Connect Protocol.

### **ssl.cipher.suites**

**Type:** list

**Default:** null

**Importance:** low

A list of cipher suites. This is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol. By default all the available cipher suites are supported.

### **ssl.client.auth**

**Type:** string

**Default:** none

**Importance:** low

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 option is set client can choose not to provide authentication information about itself
- **ssl.client.auth=none** This means client authentication is not needed.

### **ssl.endpoint.identification.algorithm**

**Type:** string

**Default:** https

**Importance:** low

The endpoint identification algorithm to validate server hostname using server certificate.

### **ssl.keymanager.algorithm**

**Type:** string

**Default:** SunX509

**Importance:** low

The algorithm used by key manager factory for SSL connections. Default value is the key manager factory algorithm configured for the Java Virtual Machine.

### **ssl.secure.random.implementation**

**Type:** string

**Default:** null

**Importance:** low

The SecureRandom PRNG implementation to use for SSL cryptography operations.

### **ssl.trustmanager.algorithm**

**Type:** string

**Default:** PKIX

**Importance:** low

The algorithm used by trust manager factory for SSL connections. Default value is the trust manager factory algorithm configured for the Java Virtual Machine.

### **status.storage.partitions**

**Type:** int

**Default:** 5

**Valid Values:** [1,...]

**Importance:** low

The number of partitions used when creating the status storage topic.

### **status.storage.replication.factor**

**Type:** short

**Default:** 3

**Valid Values:** [1,...]

**Importance:** low

Replication factor used when creating the status storage topic.

### **task.shutdown.graceful.timeout.ms**

**Type:** long

**Default:** 5000

**Importance:** low

Amount of time to wait for tasks to shutdown gracefully. This is the total amount of time, not per task. All task have shutdown triggered, then they are waited on sequentially.

## APPENDIX G. KAFKA STREAMS CONFIGURATION PARAMETERS

### **application.id**

**Type:** string

**Importance:** high

An identifier for the stream processing application. Must be unique within the Kafka cluster. It is used as 1) the default client-id prefix, 2) the group-id for membership management, 3) the changelog topic prefix.

### **bootstrap.servers**

**Type:** list

**Importance:** high

A list of host/port pairs to use for establishing the initial connection to the Kafka cluster. The client will make use of all servers irrespective of which servers are specified here for bootstrapping—this list only impacts the initial hosts used to discover the full set of servers. This list should be in the form **host1:port1,host2:port2,...**. Since these servers are just used for the initial connection to discover the full cluster membership (which may change dynamically), this list need not contain the full set of servers (you may want more than one, though, in case a server is down).

### **replication.factor**

**Type:** int

**Default:** 1

**Importance:** high

The replication factor for change log topics and repartition topics created by the stream processing application.

### **state.dir**

**Type:** string

**Default:** /tmp/kafka-streams

**Importance:** high

Directory location for state store.

### **cache.max.bytes.buffering**

**Type:** long

**Default:** 10485760

**Valid Values:** [0,...]

**Importance:** medium

Maximum number of memory bytes to be used for buffering across all threads.

### **client.id**

**Type:** string

**Default:** ""

**Importance:** medium

An ID prefix string used for the client IDs of internal consumer, producer and restore-consumer, with pattern '<client.id>-StreamThread-<threadSequenceNumber>-<consumer|producer|restore-consumer>'.  
'

### **default.deserialization.exception.handler**

**Type:** class

**Default:** org.apache.kafka.streams.errors.LogAndFailExceptionHandler

**Importance:** medium

Exception handling class that implements the

**org.apache.kafka.streams.errors.DeserializationExceptionHandler** interface.

#### default.key.serde

**Type:** class

**Default:** org.apache.kafka.common.serialization.Serdes\$ByteArraySerde

**Importance:** medium

Default serializer / deserializer class for key that implements the

**org.apache.kafka.common.serialization.Serde** interface. Note when windowed serde class is used, one needs to set the inner serde class that implements the

**org.apache.kafka.common.serialization.Serde** interface via 'default.windowed.key.serde.inner' or 'default.windowed.value.serde.inner' as well.

#### default.production.exception.handler

**Type:** class

**Default:** org.apache.kafka.streams.errors.DefaultProductionExceptionHandler

**Importance:** medium

Exception handling class that implements the

**org.apache.kafka.streams.errors.ProductionExceptionHandler** interface.

#### default.timestamp.extractor

**Type:** class

**Default:** org.apache.kafka.streams.processor.FailOnInvalidTimestamp

**Importance:** medium

Default timestamp extractor class that implements the

**org.apache.kafka.streams.processor.TimestampExtractor** interface.

#### default.value.serde

**Type:** class

**Default:** org.apache.kafka.common.serialization.Serdes\$ByteArraySerde

**Importance:** medium

Default serializer / deserializer class for value that implements the

**org.apache.kafka.common.serialization.Serde** interface. Note when windowed serde class is used, one needs to set the inner serde class that implements the

**org.apache.kafka.common.serialization.Serde** interface via 'default.windowed.key.serde.inner' or 'default.windowed.value.serde.inner' as well.

#### max.task.idle.ms

**Type:** long

**Default:** 0

**Importance:** medium

Maximum amount of time a stream task will stay idle when not all of its partition buffers contain records, to avoid potential out-of-order record processing across multiple input streams.

#### num.standby.replicas

**Type:** int

**Default:** 0

**Importance:** medium

The number of standby replicas for each task.

**num.stream.threads****Type:** int**Default:** 1**Importance:** medium

The number of threads to execute stream processing.

**processing.guarantee****Type:** string**Default:** at\_least\_once**Valid Values:** [at\_least\_once, exactly\_once]**Importance:** medium

The processing guarantee that should be used. Possible values are **at\_least\_once** (default) and **exactly\_once**. Note that exactly-once processing requires a cluster of at least three brokers by default what is the recommended setting for production; for development you can change this, by adjusting broker setting **transaction.state.log.replication.factor** and **transaction.state.log.min.isr**.

**security.protocol****Type:** string**Default:** PLAINTEXT**Importance:** medium

Protocol used to communicate with brokers. Valid values are: PLAINTEXT, SSL, SASL\_PLAINTEXT, SASL\_SSL.

**topology.optimization****Type:** string**Default:** none**Valid Values:** [none, all]**Importance:** medium

A configuration telling Kafka Streams if it should optimize the topology, disabled by default.

**application.server****Type:** string**Default:** ""**Importance:** low

A host:port pair pointing to an embedded user defined endpoint that can be used for discovering the locations of state stores within a single KafkaStreams application.

**buffered.records.per.partition****Type:** int**Default:** 1000**Importance:** low

Maximum number of records to buffer per partition.

**commit.interval.ms****Type:** long**Default:** 30000**Valid Values:** [0,...]**Importance:** low

The frequency with which to save the position of the processor. (Note, if **processing.guarantee** is set to **exactly\_once**, the default value is **100**, otherwise the default value is **30000**.)

**connections.max.idle.ms**

**Type:** long

**Default:** 540000

**Importance:** low

Close idle connections after the number of milliseconds specified by this config.

**metadata.max.age.ms**

**Type:** long

**Default:** 300000

**Valid Values:** [0,...]

**Importance:** low

The period of time in milliseconds after which we force a refresh of metadata even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions.

**metric.reporters**

**Type:** list

**Default:** ""

**Importance:** low

A list of classes to use as metrics reporters. Implementing the **org.apache.kafka.common.metrics.MetricsReporter** interface allows plugging in classes that will be notified of new metric creation. The JmxReporter is always included to register JMX statistics.

**metrics.num.samples**

**Type:** int

**Default:** 2

**Valid Values:** [1,...]

**Importance:** low

The number of samples maintained to compute metrics.

**metrics.recording.level**

**Type:** string

**Default:** INFO

**Valid Values:** [INFO, DEBUG]

**Importance:** low

The highest recording level for metrics.

**metrics.sample.window.ms**

**Type:** long

**Default:** 30000

**Valid Values:** [0,...]

**Importance:** low

The window of time a metrics sample is computed over.

**partition.grouper**

**Type:** class

**Default:** org.apache.kafka.streams.processor.DefaultPartitionGrouper

**Importance:** low

Partition grouper class that implements the **org.apache.kafka.streams.processor.PartitionGrouper** interface.

**poll.ms**

**Type:** long

**Default:** 100

**Importance:** low

The amount of time in milliseconds to block waiting for input.

### **receive.buffer.bytes**

**Type:** int

**Default:** 32768

**Valid Values:** [-1,...]

**Importance:** low

The size of the TCP receive buffer (SO\_RCVBUF) to use when reading data. If the value is -1, the OS default will be used.

### **reconnect.backoff.max.ms**

**Type:** long

**Default:** 1000

**Valid Values:** [0,...]

**Importance:** low

The maximum amount of time in milliseconds to wait when reconnecting to a broker that has repeatedly failed to connect. If provided, the backoff per host will increase exponentially for each consecutive connection failure, up to this maximum. After calculating the backoff increase, 20% random jitter is added to avoid connection storms.

### **reconnect.backoff.ms**

**Type:** long

**Default:** 50

**Valid Values:** [0,...]

**Importance:** low

The base amount of time to wait before attempting to reconnect to a given host. This avoids repeatedly connecting to a host in a tight loop. This backoff applies to all connection attempts by the client to a broker.

### **request.timeout.ms**

**Type:** int

**Default:** 40000

**Valid Values:** [0,...]

**Importance:** low

The configuration controls the maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.

### **retries**

**Type:** int

**Default:** 0

**Valid Values:** [0,...,2147483647]

**Importance:** low

Setting a value greater than zero will cause the client to resend any request that fails with a potentially transient error.

### **retry.backoff.ms**

**Type:** long

**Default:** 100



**Valid Values:** [0,...]

**Importance:** low

The amount of time to wait before attempting to retry a failed request to a given topic partition. This avoids repeatedly sending requests in a tight loop under some failure scenarios.

### **rocksdb.config.setter**

**Type:** class

**Default:** null

**Importance:** low

A Rocks DB config setter class or class name that implements the **org.apache.kafka.streams.state.RocksDBConfigSetter** interface.

### **send.buffer.bytes**

**Type:** int

**Default:** 131072

**Valid Values:** [-1,...]

**Importance:** low

The size of the TCP send buffer (SO\_SNDBUF) to use when sending data. If the value is -1, the OS default will be used.

### **state.cleanup.delay.ms**

**Type:** long

**Default:** 600000

**Importance:** low

The amount of time in milliseconds to wait before deleting state when a partition has migrated. Only state directories that have not been modified for at least **state.cleanup.delay.ms** will be removed.

### **upgrade.from**

**Type:** string

**Default:** null

**Valid Values:** [null, 0.10.0, 0.10.1, 0.10.2, 0.11.0, 1.0, 1.1]

**Importance:** low

Allows upgrading from versions 0.10.0/0.10.1/0.10.2/0.11.0/1.0/1.1 to version 1.2 (or newer) in a backward compatible way. When upgrading from 1.2 to a newer version it is not required to specify this config. Default is null. Accepted values are "0.10.0", "0.10.1", "0.10.2", "0.11.0", "1.0", "1.1" (for upgrading from the corresponding old version).

### **windowstore.changelog.additional.retention.ms**

**Type:** long

**Default:** 86400000

**Importance:** low

Added to a windows maintainMs to ensure data is not deleted from the log prematurely. Allows for clock drift. Default is 1 day.