Red Hat AMQ 7.3

Using AMQ Streams on Red Hat Enterprise Linux (RHEL)

For Use with AMQ Streams 1.2

Last Updated: 2019-06-25

Red Hat AMQ 7.3 Using AMQ Streams on Red Hat Enterprise Linux (RHEL)

For Use with AMQ Streams 1.2

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. OVERVIEW OF AMQ STREAMS 6
1.1. KEY FEATURES 6
1.2. SUPPORTED CONFIGURATIONS 7
1.3. DOCUMENT CONVENTIONS 7

CHAPTER 2. GETTING STARTED 8
2.1. AMQ STREAMS DISTRIBUTION 8
2.2. DOWNLOADING AN AMQ STREAMS ARCHIVE 8
2.3. INSTALLING AMQ STREAMS 8
2.4. DATA STORAGE CONSIDERATIONS 9
2.4.1. Apache Kafka and Zookeeper storage support 9
2.4.2. File systems 9
2.5. RUNNING SINGLE NODE AMQ STREAMS CLUSTER 10
2.6. USING THE CLUSTER 11
2.7. STOPPING THE AMQ STREAMS SERVICES 11
2.8. CONFIGURING AMQ STREAMS 12

CHAPTER 3. CONFIGURING ZOOKEEPER 14
3.1. BASIC CONFIGURATION 14
3.2. ZOOKEEPER CLUSTER CONFIGURATION 14
3.3. RUNNING MULTI-NODE ZOOKEEPER CLUSTER 16
3.4. AUTHENTICATION 17
3.4.1. Authentication with SASL 17
3.4.2. Enabling Server-to-server authentication using DIGEST-MD5 19
3.4.3. Enabling Client-to-server authentication using DIGEST-MD5 20
3.5. AUTHORIZATION 22
3.6. TLS 22
3.7. ADDITIONAL CONFIGURATION OPTIONS 22
3.8. LOGGING 22

CHAPTER 4. CONFIGURING KAFKA 23
4.1. ZOOKEEPER 23
4.2. LISTENERS 23
4.3. COMMIT LOGS 24
4.4. BROKER ID 24
4.5. RUNNING A MULTI-NODE KAFKA CLUSTER 24
4.6. ZOOKEEPER AUTHENTICATION 26
4.6.1. JAAS Configuration 26
4.6.2. Enabling Zookeeper authentication 26
4.7. ZOOKEEPER AUTHORIZATION 27
4.7.1. ACL Configuration 27
4.7.2. Enabling Zookeeper ACLs for a new Kafka cluster 28
4.7.3. Enabling Zookeeper ACLs in an existing Kafka cluster 28
4.8. ENCRYPTION AND AUTHENTICATION 29
4.8.1. Listener configuration 29
4.8.2. TLS Encryption 30
4.8.3. Enabling TLS encryption 31
4.8.4. Authentication 32
4.8.4.1. TLS client authentication 32
4.8.4.2. SASL authentication 32
4.8.5. Enabling TLS client authentication 35
4.8.6. Enabling SASL PLAIN authentication 36

4.8.7. Enabling SASL SCRAM authentication 37
4.8.8. Adding SASL SCRAM users 38
4.8.9. Deleting SASL SCRAM users 39
4.9. LOGGING 39

CHAPTER 5. TOPICS 41
5.1. PARTITIONS AND REPLICAS 41
5.2. MESSAGE RETENTION 41
5.3. TOPIC AUTO-CREATION 42
5.4. TOPIC DELETION 42
5.5. TOPIC CONFIGURATION 42
5.6. INTERNAL TOPICS 43
5.7. CREATING A TOPIC 44
5.8. LISTING AND DESCRIBING TOPICS 45
5.9. MODIFYING A TOPIC CONFIGURATION 45
5.10. DELETING A TOPIC 47

CHAPTER 6. SCALING CLUSTERS 48
6.1. SCALING KAFKA CLUSTERS 48
6.1.1. Adding brokers to a cluster 48
6.1.2. Removing brokers from the cluster 48
6.2. REASSIGNMENT OF PARTITIONS 48
6.2.1. Reassignment JSON file 49
6.2.2. Generating reassignment JSON files 49
6.2.3. Creating reassignment JSON files manually 50
6.3. REASSIGNMENT THROTTLES 50
6.4. SCALING UP A KAFKA CLUSTER 50
6.5. SCALING DOWN A KAFKA CLUSTER 52

CHAPTER 7. MONITORING YOUR CLUSTER USING JMX 54
7.1. JMX CONFIGURATION OPTIONS 54
7.2. DISABLING THE JMX AGENT 54
7.3. CONNECTING TO THE JVM FROM A DIFFERENT MACHINE 54
7.4. MONITORING USING JCONSOLE 55
7.5. IMPORTANT KAFKA BROKER METRICS 55
7.5.1. Kafka server metrics 55
7.5.2. Kafka network metrics 57
7.5.3. Kafka log metrics 59
7.5.4. Kafka controller metrics 60
7.5.5. Yammer metrics 60
7.6. PRODUCER MBEANS 61
7.6.1. MBeans matching kafka.producer:type=producer-metrics,client-id=* 61
7.6.2. MBeans matching kafka.producer:type=producer-metrics,client-id=*,node-id=* 63
7.6.3. MBeans matching kafka.producer:type=producer-topic-metrics,client-id=*,topic=* 64
7.7. CONSUMER MBEANS 64
7.7.1. MBeans matching kafka.consumer:type=consumer-metrics,client-id=* 65
7.7.2. MBeans matching kafka.consumer:type=consumer-metrics,client-id=*,node-id=* 66
7.7.3. MBeans matching kafka.consumer:type=consumer-coordinator-metrics,client-id=* 66
7.7.4. MBeans matching kafka.consumer:type=consumer-fetch-manager-metrics,client-id=* 67
7.7.5. MBeans matching kafka.consumer:type=consumer-fetch-manager-metrics,client-id=*,topic=* 68
7.7.6. MBeans matching kafka.consumer:type=consumer-fetch-manager-metrics,client-id=*,topic=*,partition=* 68
7.8. KAFKA CONNECT MBEANS 69
7.8.1. MBeans matching kafka.connect:type=connect-metrics,client-id=* 69

7.8.2. MBeans matching kafka.connect:type=connect-metrics,client-id=*,node-id=* 70
7.8.3. MBeans matching kafka.connect:type=connect-worker-metrics 71
7.8.4. MBeans matching kafka.connect:type=connect-worker-rebalance-metrics 72
7.8.5. MBeans matching kafka.connect:type=connector-metrics,connector=* 72
7.8.6. MBeans matching kafka.connect:type=connector-task-metrics,connector=*,task=* 72
7.8.7. MBeans matching kafka.connect:type=sink-task-metrics,connector=*,task=* 73
7.8.8. MBeans matching kafka.connect:type=source-task-metrics,connector=*,task=* 74
7.8.9. MBeans matching kafka.connect:type=task-error-metrics,connector=*,task=* 75
7.9. KAFKA STREAMS MBEANS 76
7.9.1. MBeans matching kafka.streams:type=stream-metrics,client-id=* 76
7.9.2. MBeans matching kafka.streams:type=stream-task-metrics,client-id=*,task-id=* 77
7.9.3. MBeans matching kafka.streams:type=stream-processor-node-metrics,client-id=*,task-id=*,processor-node-id=* 78
7.9.4. MBeans matching kafka.streams:type=stream-[store-scope]-metrics,client-id=*,task-id=*,[store-scope]-id=* 79
7.9.5. MBeans matching kafka.streams:type=stream-record-cache-metrics,client-id=*,task-id=*,record-cache-id=* 81

CHAPTER 8. KAFKA CONNECT 82
8.1. KAFKA CONNECT IN STANDALONE MODE 82
8.1.1. Configuring Kafka Connect in standalone mode 82
8.1.2. Configuring connectors in Kafka Connect in standalone mode 83
8.1.3. Running Kafka Connect in standalone mode 83
8.2. KAFKA CONNECT IN DISTRIBUTED MODE 84
8.2.1. Configuring Kafka Connect in distributed mode 84
8.2.2. Configuring connectors in distributed Kafka Connect 85
8.2.3. Running distributed Kafka Connect 86
8.2.4. Creating connectors 87
8.2.5. Deleting connectors 88
8.3. CONNECTOR PLUG-INS 88
8.4. ADDING CONNECTOR PLUGINS 89

CHAPTER 9. KAFKA CLIENTS 90
9.1. ADDING KAFKA CLIENTS AS A DEPENDENCY TO YOUR MAVEN PROJECT 90

CHAPTER 10. KAFKA STREAMS API OVERVIEW 92
10.1. ADDING THE KAFKA STREAMS API AS A DEPENDENCY TO YOUR MAVEN PROJECT 92

CHAPTER 11. USING THE AMQ STREAMS KAFKA BRIDGE 94
11.1. OVERVIEW OF THE AMQ STREAMS KAFKA BRIDGE 94
11.2. REQUESTS TO THE AMQ STREAMS KAFKA BRIDGE 94
11.2.1. Authentication and encryption 94
11.2.2. Data formats and headers 95
11.2.2.1. Content Type headers 95
11.2.2.2. Embedded data format 95
11.2.2.3. Accept headers 96
11.3. DOWNLOADING AN AMQ STREAMS ARCHIVE 96
11.4. CONFIGURING AMQ STREAMS KAFKA BRIDGE PROPERTIES 96
11.5. INSTALLING THE AMQ STREAMS KAFKA BRIDGE ON RED HAT ENTERPRISE LINUX 97
11.6. AMQ STREAMS KAFKA BRIDGE API RESOURCES 98

CHAPTER 12. AMQ STREAMS AND KAFKA UPGRADES 99
12.1. UPGRADE PREREQUISITES 99
12.2. UPGRADE PROCESS 99
12.3. KAFKA VERSIONS 99

12.4. UPGRADING TO AMQ STREAMS 1.2 100
12.4.1. Upgrading Zookeeper 100
12.4.2. Upgrading Kafka brokers 101
12.4.3. Upgrading Kafka Connect 102
12.5. UPGRADING KAFKA 104
12.5.1. Upgrading Kafka brokers to use the new inter-broker protocol version 104
12.5.2. Strategies for upgrading clients 105
12.5.3. Upgrading client applications to the new Kafka version 107
12.5.4. Upgrading Kafka brokers to use the new message format version 107

APPENDIX A. BROKER CONFIGURATION PARAMETERS 109

APPENDIX B. TOPIC CONFIGURATION PARAMETERS 139

APPENDIX C. CONSUMER CONFIGURATION PARAMETERS 144

APPENDIX D. PRODUCER CONFIGURATION PARAMETERS 155

APPENDIX E. ADMIN CLIENT CONFIGURATION PARAMETERS 166

APPENDIX F. KAFKA CONNECT CONFIGURATION PARAMETERS 173

APPENDIX G. KAFKA STREAMS CONFIGURATION PARAMETERS 185

APPENDIX H. USING YOUR SUBSCRIPTION 191
Accessing Your Account 191
Activating a Subscription 191
Downloading Zip and Tar Files 191

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. It consists of the following main components:

Zookeeper

Service for highly reliable distributed coordination.

Kafka Broker

Messaging broker responsible for delivering records from producing clients to consuming clients.

Kafka Connect

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

Kafka Consumer and Producer APIs

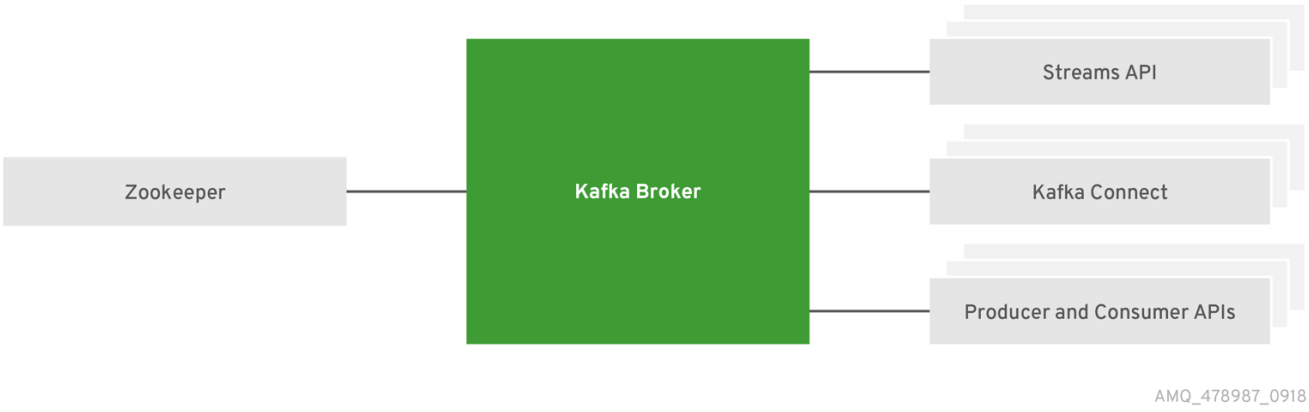
Java based APIs for producing and consuming messages to and from Kafka brokers.

Kafka Streams API

API for writing *stream processor* applications.

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. Example Architecture diagram of AMQ Streams



1.1. KEY FEATURES

- Scalability and performance
  - Designed for horizontal scalability
- Message ordering guarantee
  - At partition level
- Message rewind/replay
  - "Long term" storage
  - Allows to reconstruct application state by replaying the messages
  - Combined with compacted topics allows to use Kafka as key-value store

1.2. 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 [a]
OracleJDK	1.8
IBM JDK	1.8

[a] 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

1.3. 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 all AMQ Streams components:

- Apache Zookeeper
- Apache Kafka
- Apache Kafka Connect
- Apache Kafka Mirror Maker

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 1.2, see AMQ Streams and Kafka upgrades.

Prerequisites

- Download the installation archive.
- Review the Section 1.2, “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.

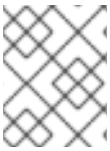
Choose local storage when possible. If local storage is not available, you can use a Storage Area Network (SAN) accessed by a protocol such as Fibre Channel or iSCSI.

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 SINGLE NODE AMQ STREAMS CLUSTER

This procedure will show you how to run a basic AMQ Streams cluster consisting of single Zookeeper and single Apache Kafka node both running on the same host. It is using the default configuration files for both Zookeeper and Kafka.

WARNING

Single node AMQ Streams cluster does not provide realibility 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
```

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
```

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

Prerequisites

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

Procedure

1. Start the Kafka console producer.

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

For example:

```
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.
4. Press Ctrl+C to exit the Kafka console producer.
5. Start the message receiver.

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

For example:

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

6. Confirm that you see the incoming messages in the consumer console.
7. Press Crtl+C to exit the Kafka console 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

- 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 log and snapshots of its in-memory database. This should be set to the **/var/lib/zookeeper/** directory created during installation.

clientPort

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

An example zookeeper configuration file **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.

```
timeTick=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 as ensembles.

Zookeeper clusters usually consist of an odd number of nodes. Zookeeper requires a majority of the nodes in the cluster should be up and running. For example, a cluster with three nodes, at least two of them must be up and running. This means it can tolerate one node being down. A cluster consisting of five nodes, at least three nodes must be available. This means it can tolerate two nodes being down. 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.

The different Zookeeper nodes should be ideally placed into different data centers or network segments. Increasing the number of Zookeeper nodes increases the workload spent on cluster synchronization. For most Kafka use cases Zookeeper cluster with 3, 5 or 7 nodes should be fully sufficient.

WARNING

Zookeeper cluster with 3 nodes can tolerate only 1 unavailable node. This means that when 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).

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/
clientPort=2181
initLimit=5
syncLimit=2

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

Each node in the Zookeeper cluster has to be assigned with a unique **ID**. Each node's **ID** has to be configured in **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.

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.
   - 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/
clientPort=2181
initLimit=5
syncLimit=2

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

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 the 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:
 /0:0:0:0:0:0:0:1:59726[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 **EXTRA_ARGS** environment variable to pass the JAAS configuration file to the Zookeeper server as a Java property:

```
su - kafka
export EXTRA_ARGS="-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 **EXTRA_ARGS** environment variable to pass the JAAS configuration file to the Zookeeper server as a Java property:

```
su - kafka
export EXTRA_ARGS="-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
```

3. Restart all Zookeeper nodes one by one. To pass the JAAS configuration to Zookeeper, use the **EXTRA_ARGS** environment variable.

```
su - kafka
export EXTRA_ARGS="-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

1. 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** will have automatically administrator priviledges. 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";
};
```

2. 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
```

3. Restart all Zookeeper nodes one by one. To pass the JAAS configuration to Zookeeper, use the **EXTRA_ARGS** environment variable.

```
su - kafka
export EXTRA_ARGS="-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 4.7, “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 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 Zookeeper documentation.

3.8. LOGGING

Zookeeper is using *log4j* as their logging infrastructure. Logging configuration is by default read from the **log4j.propeties** 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
```

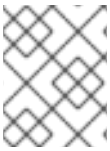
When a Kafka client wants to connect to a Kafka cluster, it first connects to a *bootstrap server*. The *bootstrap server* is one of the cluster nodes. It will provide the client with a list of all other brokers which are part of the cluster and the client will connect to them individually. By default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

When authentication is enabled between Kafka and Zookeeper, Kafka can be configured to automatically protect all its records with Access Control List (ACL) rules which will allow only the Kafka user to change the data. All other users will have read-only access.

4.7.1. ACL Configuration

Enforcement of ACL rules is controlled by the **zookeeper.set.acl** property in the **config/server.properties** Kafka configuration file and is disabled by default. To enabled the ACL protection set **zookeeper.set.acl** to **true**:

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

Kafka will set the ACL rules only for newly created Zookeeper **znodes**. When the ACLs are only enabled after the first start of the cluster, the tool **zookeeper-security-migration.sh** can be used to set ACLs on all existing **znodes**.

The data stored in Zookeeper includes information such as topic names and their configuration. The Zookeeper database also contains the salted and hashed user credentials when SASL SCRAM authentication is used. But it does not include any records sent and received using Kafka. Kafka, in

general, considers the data stored in Zookeeper as non-confidential. In case these data are considered confidential (for example because topic names contain customer identification) the only way how to protect them is by isolating Zookeeper on the network level and allowing access only to Kafka brokers.

4.7.2. 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 already running cluster, see Section 4.7.3, "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 broker 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.

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 more information about running a Kafka cluster, see Section 4.5, "Running a multi-node Kafka cluster".

4.7.3. Enabling Zookeeper ACLs in an existing Kafka cluster

The **zookeeper-security-migration.sh** tool can to be used 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
```

4.8. ENCRYPTION AND AUTHENTICATION

Kafka supports TLS for encrypting the communication with Kafka clients. Additionally, it supports two types of authentication:

- TLS client authentication based on X.509 certificates
- SASL Authentication based on a username and password

4.8.1. 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.

4.8.2. TLS Encryption

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*.

4.8.3. 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*

4.8.4. Authentication

Kafka supports two methods of authentication. On all connections, authentication using one of the supported SASL (Simple Authentication and Security Layer) mechanisms can be used. On encrypted connections, TLS client authentication based on X.509 certificates can be used.

4.8.4.1. 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**.

4.8.4.2. 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.config"; 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.

4.8.5. 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*

4.8.6. 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*

4.8.7. 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 4.8.8, "Adding SASL SCRAM users".
- For more information about deleting SASL SCRAM users, see Section 4.8.9, "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*

4.8.8. 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*

4.8.9. 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*

4.9. LOGGING

Kafka brokers use Log4j as their logging infrastructure. Logging configuration is by default read from the **log4j.propeties** 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 parition 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 sent to or received from from a topic which 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 **1000012**.

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.

5. 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.

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

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

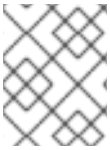
7. Confirm that the Kafka broker is stopped.

```
jcmd | grep kafka
```

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 of this 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 various 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.

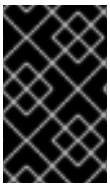
Procedure

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	**org.apache.zookeeper.server.quorum.QuorumPeerMain**
Kafka broker	**kafka.Kafka**
Kafka Connect standalone	**org.apache.kafka.connect.cli.ConnectStandalone**
Kafka Connect distributed	**org.apache.kafka.connect.cli.ConnectDistributed**
A Kafka producer, consumer, or Streams application	The name of the class containing the **main** 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	**kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec**	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	**kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec**	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	**kafka.server:type=BrokerTopicMetrics,name=ReplicationBytesInPerSec**	The rate at which data sent from other brokers is consumed by the follower broker.	N/A
Bytes out per second	**kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec**	The rate at which data is fetched and read from the broker by consumers.	N/A
Replication bytes out per second	**kafka.server:type=BrokerTopicMetrics,name=ReplicationBytesOutPerSec**	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	**kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions**	The number of partitions that have not been fully replicated in the follower replicas.	Zero
Under minimum ISR partition count	**kafka.server:type=ReplicaManager,name=UnderMinIsrPartitionCount**	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	**kafka.server:type=ReplicaManager,name=PartitionCount**	The number of partitions in the broker.	Approximately even when compared with the other brokers.
Leader count	**kafka.server:type=ReplicaManager,name=LeaderCount**	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	**kafka.server:type=ReplicaManager,name=IsrShrinksPerSec**	The rate at which the number of ISRs in the broker decreases	Zero
ISR expands per second	**kafka.server:type=ReplicaManager,name=IsrExpandsPerSec**	The rate at which the number of ISRs in the broker increases.	Zero
Maximum lag	**kafka.server:type=ReplicaFetcherManager,name=MaxLag,clientId=Replica**	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	**kafka.server:type=DelayedOperationPurgatory,name=PurgatorySize,delayedOperation=Produce**	The number of send requests in the producer purgatory.	N/A
Requests in fetch purgatory	**kafka.server:type=DelayedOperationPurgatory,name=PurgatorySize,delayedOperation=Fetch**	The number of fetch requests in the fetch purgatory.	N/A
Request handler average idle percent	**kafka.server:type=KafkaRequestHandlerPool,name=RequestHandlerAvgIdlePercent**	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)	**kafka.server:type=Request**	The number of requests that are exempt from throttling.	N/A
Zookeeper request latency in milliseconds	**kafka.server:type=ZooKeeperClientMetrics,name=ZooKeeperRequestLatencyMs**	The latency for ZooKeeper requests from the broker, in milliseconds.	N/A
Zookeeper session state	**kafka.server:type=SessionExpireListener,name=SessionState**	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	**kafka.network:type=RequestMetrics,name=RequestsPerSec,request={Produce|FetchConsumer|FetchFollower}**	The total number of requests made for the request type per second. The **Produce**, **FetchConsumer**, and **FetchFollower** request types each have their own MBeans.	N/A
Request bytes (request size in bytes)	**kafka.network:type=RequestMetrics,name=RequestBytes,request=([-.\w]+)**	The size of requests, in bytes, made for the request type identified by the **request** property of the MBean name. Separate MBeans for all available request types are listed under the **RequestBytes** node.	N/A
Temporary memory size in bytes	**kafka.network:type=RequestMetrics,name=TemporaryMemoryBytes,request={Produce|Fetch}**	The amount of temporary memory used for converting message formats and decompressing messages.	N/A
Message conversions time	**kafka.network:type=RequestMetrics,name=MessageConversionsTimeMs,request={Produce|Fetch}**	Time, in milliseconds, spent on converting message formats.	N/A
Total request time in milliseconds	**kafka.network:type=RequestMetrics,name=TotalTimeMs,request={Produce|FetchConsumer|FetchFollower}**	Total time, in milliseconds, spent processing requests.	N/A
Request queue time in milliseconds	**kafka.network:type=RequestMetrics,name=RequestQueueTimeMs,request={Produce|FetchConsumer|FetchFollower}**	The time, in milliseconds, that a request currently spends in the queue for the request type given in the **request** property.	N/A

Metric	MBean	Description	Expected value
Local time (leader local processing time) in milliseconds	**kafka.network:type=RequestMetrics,name=LocalTimeMs,request={Produce|FetchConsumer|FetchFollower}**	The time taken, in milliseconds, for the leader to process the request.	N/A
Remote time (leader remote processing time) in milliseconds	**kafka.network:type=RequestMetrics,name=RemoteTimeMs,request={Produce|FetchConsumer|FetchFollower}**	The length of time, in milliseconds, that the request waits for the follower. Separate MBeans for all available request types are listed under the **RemoteTimeMs** node.	N/A
Response queue time in milliseconds	**kafka.network:type=RequestMetrics,name=ResponseQueueTimeMs,request={Produce|FetchConsumer|FetchFollower}**	The length of time, in milliseconds, that the request waits in the response queue.	N/A
Response send time in milliseconds	**kafka.network:type=RequestMetrics,name=ResponseSendTimeMs,request={Produce|FetchConsumer|FetchFollower}**	The time taken, in milliseconds, to send the response.	N/A
Network processor average idle percent	**kafka.network:type=SocketServer,name=NetworkProcessorAvgIdlePercent**	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	**kafka.log:type=LogFlushStats,name=LogFlushRateAndTimeMs**	The rate at which log data is written to disk, in milliseconds.	N/A

Metric	MBean	Description	Expected Value
Offline log directory count	**kafka.log:type=LogManager,name=OfflineLogDirectoryCount**	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	**kafka.controller:type=KafkaController,name=ActiveControllerCount**	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	**kafka.controller:type=ControllerStats,name=LeaderElectionRateAndTimeMs**	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
- 75/95/98/99/99.9th 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/polled (before transformation) by this task belonging to the named source connector in this worker.
source-record-poll-total	The total number of records produced/polled (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 mininum 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. Connetors 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. 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.

9.1. 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>2.2.1.redhat-00002</version>
        </dependency>
    </dependencies>

    <!-- ... -->
</project>
```

3. Build your Maven project.

CHAPTER 10. 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.

10.1. 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>2.2.1.redhat-00002</version>
```

```
        </dependency>
    </dependencies>

    <!-- ... -->
</project>
```

3. Build your Maven project.

CHAPTER 11. USING THE AMQ STREAMS KAFKA BRIDGE

This chapter provides an overview of the AMQ Streams Kafka Bridge and helps you get started using the REST API.

NOTE

For the full list of REST API endpoints and descriptions, including example requests and responses, see Kafka Bridge API reference .

11.1. OVERVIEW OF THE AMQ STREAMS KAFKA BRIDGE

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 these 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.
- 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.

For more information on configuring the host and port for the **KafkaBridge** resource, see Section 11.4, “Configuring AMQ Streams Kafka Bridge properties”.

11.2. REQUESTS TO THE AMQ STREAMS KAFKA BRIDGE

11.2.1. 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.

11.2.2. Data formats and headers

Specify data formats and HTTP headers to ensure valid requests are submitted to the Kafka Bridge.

11.2.2.1. Content Type headers

API request and response bodies are always encoded as JSON.

- When performing consumer operations, **POST** requests must provide the following **Content-Type** header:

```
Content-Type: application/vnd.kafka.v2+json
```

- When performing producer operations, **POST** requests must provide the following **Content-Type** header specifying the **embedded data format** of the consumer, either **json** or **binary**, as shown in the following table.

Embedded data format	Content-Type header
JSON	**Content-Type: application/vnd.kafka.json.v2+json**
Binary	**Content-Type: application/vnd.kafka.binary.v2+json**

You set the embedded data format when creating a consumer using the **consumers/*groupid*** endpoint —for more information, see the next section.

11.2.2.2. 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 and 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, for example:

```
{
  "name": "my-consumer",
  "format": "binary", (1)
...
}
```

(1) A binary embedded data format.

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": "ZWR3YXJkdGhldGhyZWVsZWdnZWRjYXQ="
    },
  ]
}
```

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**.

11.2.2.3. Accept headers

After creating a consumer, subsequent GET requests must provide an **Accept** header in the following format:

```
Accept: application/vnd.kafka.embedded-data-format.v2+json
```

Where the **embedded-data-format** is the embedded data format of the consumer: 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
```

11.3. DOWNLOADING AN AMQ STREAMS ARCHIVE

A zipped 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.

11.4. CONFIGURING AMQ STREAMS 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 producer and consumer, 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.

For more on

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.

   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
```

11.5. INSTALLING THE AMQ STREAMS KAFKA BRIDGE ON RED HAT ENTERPRISE LINUX

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
```

11.6. AMQ STREAMS KAFKA BRIDGE API RESOURCES

For the full list of REST API endpoints and descriptions, including example requests and responses, see Kafka Bridge API reference .

CHAPTER 12. 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.

12.1. 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 1.2 on Red Hat Enterprise Linux Release Notes.

12.2. 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 1.2
2. Upgrade all Kafka brokers and client applications to the latest Kafka version
   - Upgrading Kafka

12.3. KAFKA VERSIONS

AMQ Streams is based on a specific version of Apache Kafka.

AMQ Streams version	Kafka version
1.2	2.2.1

Kafka's log message format version and inter-broker protocol version specify the version of the format of messages appended to broker logs and the version of protocol used between brokers 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
2.1.1	2.1	2.1	3.4.13

Kafka version	Interbroker protocol version	Log message format version	Zookeeper version
2.2.1	2.2	2.2	3.4.14

Although Kafka versions may use the same version of Zookeeper, it is recommended that you update your Zookeeper cluster to use the newest Zookeeper binaries before proceeding with the main AMQ Streams upgrade.

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 at the broker level by the **log.message.format.version**.

12.4. UPGRADING TO AMQ STREAMS 1.2

The steps to upgrade your deployment to use AMQ Streams 1.2 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.

12.4.1. Upgrading Zookeeper

This procedure describes how to upgrade Zookeeper 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 for this release 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 -rf /tmp/kafka
```

6. Restart Zookeeper:

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

12.4.2. 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 2.2.1 by adding or updating the following properties in the file:

```
inter.broker.protocol.version=2.1
log.message.format.version=2.1
```

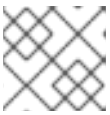
   This configures the Kafka broker to process data using the previous inter-broker protocol (**2.1**) and message format versions.

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.

12.4.3. 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.

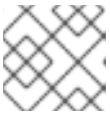
Prerequisites

- 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:

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. 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

12.5. 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 12.5.1, "Upgrading Kafka brokers to use the new inter-broker protocol version"
2. Section 12.5.3, "Upgrading client applications to the new Kafka version"
3. Section 12.5.4, "Upgrading Kafka brokers to use the new message format version"

Additional resources

- Section 12.4, "Upgrading to AMQ Streams 1.2"

12.5.1. 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 1.2
- 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 **2.2**.

```
inter.broker.protocol.version=2.2
```

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.

12.5.2. Strategies for upgrading clients

The best approach to upgrading your client applications depends on your particular circumstances. Client applications might include producers, consumers, Kafka Connect, and Kafka MirrorMaker.

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.

Another aspect to consider is that once new-version messages have been published to a topic then they will be visible to consumers. This is 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.

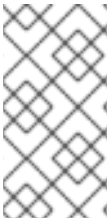
Per-topic consumers first, with down conversion

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.

12.5.3. 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 1.2.

The procedure is based on the "per-topic consumers first, with down conversion" approach outlined in Strategies for upgrading clients.

Client applications might include producers, consumers, Kafka Connect, and MirrorMaker.

Prerequisites

- You have updated the Zookeeper binaries.
- You have upgraded all Kafka brokers to AMQ Streams 1.2.
- 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 **2.1**.

```
bin/kafka-configs.sh --zookeeper <ZookeeperAddress> --entity-type topics --entity-name
<TopicName> --alter --add-config message.format.version=2.1
```

2. Upgrade all the consuming and producing applications for the topic.
3. Verify that the upgraded applications function correctly.
4. Change the topic's **message.format.version** configuration option to **2.2**.

```
bin/kafka-configs.sh --zookeeper <ZookeeperAddress> --entity-type topics --entity-name
<TopicName> --alter --add-config message.format.version=2.2
```

Additional resources

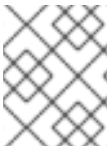
- Strategies for upgrading clients

12.5.4. 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 1.2, 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 1.2.
- 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 **2.2**.

```
log.message.format.version=2.2
```

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.

APPENDIX A. BROKER CONFIGURATION PARAMETERS

zookeeper.connect

Type: string
Importance: high
Dynamic update: read-only
Zookeeper host string.

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'). It additionally accepts 'uncompressed' which is equivalent to no compression; and 'producer' which means retain the original compression codec set by the producer.

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.

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.

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
Offsets older than this retention period will be discarded.

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 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 sever 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 sever 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.

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.

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 seconds 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: 300000
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.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: 2.0-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.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.

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: 2.0-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.per.ip

Type: int
Default: 2147483647
Valid Values: [0,...]
Importance: medium
Dynamic update: read-only

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.

max.connections.per.ip.overrides

Type: string
Default: ""
Importance: medium
Dynamic update: read-only
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 name will be 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_authz[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 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 key manager factory for SSL connections. 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.

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.

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 configurationdoes 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.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". 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 #compaction[log compaction] on the topic.

compression.type

Type: string
Default: producer
Valid Values: [uncompressed, snappy, lz4, 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). 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 #compaction[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 #topicconfigs[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.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: 2.0-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 #compaction[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.

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
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 configurationdoes 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 value
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: ""
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.

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.

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 records from internal topics (such as offsets) should be exposed to the consumer. If set to **true** the only way to receive records from an internal topic is subscribing to it.

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 a 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.

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</mode> 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 value
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.

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 value
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 value
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 value
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**, or **lz4**. 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: 0
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.

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.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.

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 value
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 value
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 transactions 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**.

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.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.

connections.max.idle.ms

Type: long
Default: 540000
Importance: medium
Close idle connections after the number of milliseconds specified by this config.

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.

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.

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.

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**.

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
The maximum number of records to buffer per partition.

commit.interval.ms

Type: long
Default: 30000
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: [0,...]
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: [0,...]
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.

APPENDIX H. USING YOUR SUBSCRIPTION

AMQ Streams is provided through a software subscription. To manage your subscriptions, access your account at the Red Hat Customer Portal.

Accessing Your Account

1. Go to access.redhat.com.
2. If you do not already have an account, create one.
3. Log in to your account.

Activating a Subscription

1. Go to access.redhat.com.
2. Navigate to **My Subscriptions**.
3. Navigate to **Activate a subscription** and enter your 16-digit activation number.

Downloading Zip and Tar Files

To access zip or tar files, use the customer portal to find the relevant files for download.

1. Open a browser and log in to the Red Hat Customer Portal **Product Downloads** page at access.redhat.com/downloads.
2. Locate the **Red Hat AMQ Streams** entries in the **JBOSS INTEGRATION AND AUTOMATION** category.
3. Select the desired AMQ Streams product. The **Software Downloads** page opens.
4. Click the **Download** link for your component.

Revised on 2019-06-25 06:53:29 UTC