

## Securing Apache Kafka

Date published: 2019-12-18

Date modified: 2021-06-08



# Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

# Contents

<b>Channel encryption.....</b>	<b>4</b>
Configure TLS/SSL encryption for Kafka brokers.....	4
Configure TLS/SSL encryption for Kafka clients.....	5
Configuring TLS/SSL encryption for Kafka MirrorMaker.....	6
Configure Zookeeper TLS/SSL support for Kafka.....	7
 <b>Authentication.....</b>	 <b>8</b>
TLS/SSL client authentication.....	8
Configure TLS/SSL client authentication for Kafka brokers.....	8
Configure TLS/SSL authentication for Kafka clients.....	9
Principal name mapping.....	11
Enable Kerberos authentication for Kafka.....	11
Delegation token based authentication.....	13
Enable or disable authentication with delegation tokens.....	14
Manage individual delegation tokens.....	14
Rotate the master key/secret.....	15
Client authentication using delegation tokens.....	16
Kafka security hardening with Zookeeper ACLs.....	17
LDAP authentication.....	19
Configure LDAP authentication for Kafka brokers.....	19
Configure Kafka LDAP authentication for Kafka clients.....	20
PAM authentication.....	22
Configure PAM authentication for Kafka brokers.....	22
Configure PAM authentication for Kafka clients.....	22
 <b>Authorization.....</b>	 <b>23</b>
Ranger.....	23
Enable authorization in Kafka with Ranger.....	24
Configure the resource-based Ranger service used for authorization.....	24
 <b>Governance.....</b>	 <b>25</b>
Configuring the Atlas hook in Kafka.....	25
 <b>Inter-broker security.....</b>	 <b>27</b>
 <b>Configuring multiple listeners.....</b>	 <b>27</b>

## Channel encryption

Kafka supports client to broker and inter-broker TLS/SSL encrypted communications. Configuring TLS/SSL encryption for a Kafka deployment involves configuring both clients and brokers. In addition to this, Kafka also supports TLS/SSL communication with Zookeeper.

### Configure TLS/SSL encryption for Kafka brokers

Kafka supports TLS/SSL encrypted communication with both brokers and clients. To enable and configure TLS/SSL, you need to enable TLS/SSL for the brokers and enter key and truststore related information.

#### About this task

The following list of steps walks you through the configuration required to set up TLS/SSL encryption for Kafka brokers. It lists all mandatory configuration properties as well as a number of optional properties that you can configure.

Kafka brokers support multiple key and truststore types. The following instructions, however, do not provide details regarding how the key or truststore type used by the brokers is configured. This is because the store type is not configured at a broker level. Instead, it is configured on Cloudera Manager's central security page by going to AdministrationSettingsJava Keystore Type.

#### Before you begin

- Generate or acquire a key and truststore for your brokers which contain all necessary keys and certificates.
- Note down the locations and passwords for the key and truststores. You will need to provide these during configuration.

#### Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to Configuration.
3. Find and configure the following properties based on your cluster and requirements.

Cloudera Manager Property	Description
Enable TLS/SSL for Kafka Broker	Enables or disables TLS/SSL communication between clients and the Kafka broker.
Kafka Broker TLS/SSL Server JKS Keystore File Location	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when the Kafka broker is acting as a TLS/SSL server.
Kafka Broker TLS/SSL Server JKS Keystore File Password	The password for the Kafka broker keystore file.
Kafka Broker TLS/SSL Server JKS Keystore Key Password	The password that protects the private key contained in the keystore used when the Kafka broker is acting as a TLS/SSL server.
Kafka Broker TLS/SSL Client Trust Store File	The location on disk of the truststore used to confirm the authenticity of TLS/SSL servers that the Kafka broker might connect to. This is used when the Kafka broker is the client in a TLS/SSL connection. This truststore must contain the certificate(s) that signed the certificate(s) of the connected service(s). If this parameter is not provided, the default list of known certificate authorities is used instead.

Cloudera Manager Property	Description
Kafka Broker TLS/SSL Client Trust Store Password	The password for the Kafka broker truststore file. This password is not mandatory to access the truststore; this field can be left blank. This password provides optional integrity checking of the file. The contents of truststores are certificates, and certificates are public information.

4. (Optional) Configure additional properties.

If required, additional TLS/SSL related properties can be configured with the Kafka Broker Advanced Configuration Snippet (Safety Valve) for `kafka.properties` property. The following are some of the most commonly used optional properties:

Kafka Broker Property	Description
<code>ssl.provider</code>	The name of the security provider used for TLS/SSL connections. Default is the default security provider of the JVM.
<code>ssl.cipher.suites</code>	A cipher suite is a named combination of authentication, encryption, MAC, and a key exchange algorithm used to negotiate the security settings for a network connection using the TLS/SSL network protocol.
<code>ssl.enabled.protocols</code>	List of enabled protocols, for example, TLSv1.2,TLSv1.1,TLSv1. Should contain at least one protocol.

5. Click Save Changes.

6. Restart the Kafka service.

## Results

Kafka brokers are configured for TLS/SSL encryption.

## What to do next

Configure your clients for TLS/SSL encryption. Alternatively, you can also continue with configuring TLS/SSL authentication for the brokers.

## Related Information

[Configure TLS/SSL client authentication for Kafka brokers](#)

## Configure TLS/SSL encryption for Kafka clients

Kafka supports TLS/SSL encrypted communication with both brokers and clients. Client configuration is done by setting the relevant security-related properties for the client.

## About this task

The following steps demonstrate configuration for the console consumer or producer. If you are configuring a custom developed client, see *Java client security examples* or *.Net client security examples* for code examples.

## Before you begin

- Generate or acquire a truststore containing the certificate of the Certificate Authority that issued the broker certificates.
- Note down the location and password for the truststore. You will need to provide these during configuration.

## Procedure

1. Create a `.properties` file.

In this example the file is named `client.properties`.

2. Add the mandatory properties to the file.

The following configuration example contains all mandatory properties:

```
security.protocol=SSL
ssl.truststore.location= [***PATH TO CLIENT TRUSTSTORE***]
ssl.truststore.password=[***PASSWORD***]
```

3. (Optional) Add additional security properties to the file.

Depending on your requirements and broker configuration, additional configuration properties might also be needed. The following are some of the most commonly used optional properties:

- ssl.provider
- ssl.cipher.suites
- ssl.enabled.protocols
- ssl.truststore.type
- ssl.keystore.type

4. Run the client.

#### Console Producer

```
kafka-console-producer --bootstrap-server [***HOST1:PORT1***] --t
opic [***TOPIC***] --producer.config client.properties
```

#### Console Consumer

```
kafka-console-consumer --bootstrap-server [***HOST1:PORT1***] --t
opic [***TOPIC***] --consumer.config client.properties
```

### Results

Kafka clients are configured for TLS/SSL encryption.

### Related Information

[Java client security examples](#)

[.Net client security examples](#)

## Configuring TLS/SSL encryption for Kafka MirrorMaker

The Kafka MirrorMaker role supports TLS/SSL encrypted communication with Kafka brokers. To enable and configure TLS/SSL, you need to enable TLS/SSL for the MirrorMaker role, enter key and truststore related information, and specify the client authentication used by the source and destination Kafka clusters.

### Before you begin

- Generate or acquire a key and truststore for the MirrorMaker role which contain all necessary keys and certificates.
- Note down the locations and passwords for the key and truststores. You will need to provide these during configuration.

### Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to Configuration.

- Find and configure the following properties based on your cluster and requirements.

**Table 1:**

Cloudera Manager Property	Description
Enable TLS/SSL for Kafka MirrorMaker ssl_enabled	Encrypt communication between clients and Kafka MirrorMaker using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
Kafka MirrorMaker TLS/SSL Server JKS Keystore File Location ssl_server_keystore_location	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Kafka MirrorMaker is acting as a TLS/SSL server.
Kafka MirrorMaker TLS/SSL Server JKS Keystore File Password ssl_server_keystore_password	The password for the Kafka MirrorMaker keystore file.
Kafka MirrorMaker TLS/SSL Server JKS Keystore Key Password ssl_server_keystore_keypassword	The password that protects the private key contained in the JKS keystore used when Kafka MirrorMaker is acting as a TLS/SSL server.
Kafka MirrorMaker TLS/SSL Trust Store File ssl_client_truststore_location	The location on disk of the trust store, used to confirm the authenticity of TLS/SSL servers that Kafka MirrorMaker might connect to. This trust store must contain the certificate(s) used to sign the service(s) connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
Kafka MirrorMaker TLS/SSL Trust Store Password ssl_client_truststore_password	The password for the Kafka MirrorMaker TLS/SSL Trust Store File. This password is not required to access the trust store; this field can be left blank. This password provides optional integrity checking of the file. The contents of trust stores are certificates, and certificates are public information.
Source Kafka Cluster's Client Auth source.ssl.client.auth	Only required if the source Kafka cluster requires client authentication.
Destination Kafka Cluster's Client Auth destination.ssl.client.auth	Only required if destination Kafka cluster requires client authentication.

- Click Save Changes.
- Restart the Kafka service.

## Results

The Kafka MirrorMaker role is configured for TLS/SSL encryption.

## Configure Zookeeper TLS/SSL support for Kafka

Learn how to configure TLS/SSL communication between Kafka and Zookeeper.

### About this task

You can configure Kafka to connect to and communicate with Zookeeper through a secure TLS/SSL channel. The feature can be enabled or disabled with the Enable Secure Connection to ZooKeeper property. This property is set to true by default, however, it only takes effect if the Enable TLS/SSL for ZooKeeper property is also enabled for the dependent ZooKeeper service.

### Before you begin

If you want to enable secure connections to Zookeeper, make sure that the Enable TLS/SSL for ZooKeeper property is enabled for the dependent ZooKeeper service. For more information, see [Configure ZooKeeper TLS/SSL using Cloudera Manager](#).

### Procedure

1. In Cloudera Manager, select the Kafka service.
2. Select Configuration and find the Enable Secure Connection to ZooKeeper property.
3. Enable or disable Zookeeper TLS/SSL support for Kafka for all required services by selecting or clearing the checkbox next to the name of the service.
4. Enter a Reason for change, and click Save Changes to commit the changes.
5. Restart the Kafka service.

### Results

Zookeeper TLS/SSL support for Kafka is enabled or disabled for the selected Kafka services. If the feature is enabled, the selected Kafka services communicate with their dependent Zookeeper service through a secure TLS/SSL channel.

## Authentication

### TLS/SSL client authentication

Kafka supports TLS/SSL authentication for its clients. Configuring TLS/SSL authentication for a Kafka deployment involves enabling TLS/SSL encryption for the brokers and then configuring both clients and brokers for TLS/SSL authentication.

### Configure TLS/SSL client authentication for Kafka brokers

Kafka supports TLS/SSL authentication (two-way authentication). To enable and configure TLS/SSL client authentication, you need to enable TLS/SSL encryption and set client authentication to be required by the brokers.

#### About this task

TLS/SSL authentication for Kafka brokers can be configured with the SSL Client Authentication property. The property has three valid values, required, requested, and none. If set to required, all clients connecting to the broker will be required to authenticate with TLS/SSL. If set to requested, authentication will be requested by the broker, but clients without certificates will still be able to connect. If set to none, no SSL authentication is required.

The authenticated user's identity is determined by how the SSL Client Authentication property is configured and whether a certificate is presented. The following table collects the possible outcomes:

**Table 2: SSL Client Authentication options and outcomes**

SSL Client Authentication	Client certificate is presented	Client certificate is not presented
required	<ul style="list-style-type: none"> <li>• authentication: successful</li> <li>• authenticated user: <i>[***SUBJECT FROM THE CERTIFICATE***]</i></li> </ul>	<ul style="list-style-type: none"> <li>• authentication: fails</li> </ul>
requested	<ul style="list-style-type: none"> <li>• authentication: successful</li> <li>• authenticated user: <i>[***SUBJECT FROM THE CERTIFICATE***]</i></li> </ul>	<ul style="list-style-type: none"> <li>• authentication: successful</li> <li>• authenticated user: ANONYMOUS</li> </ul>
none	<ul style="list-style-type: none"> <li>• authentication: successful</li> <li>• authenticated user: ANONYMOUS</li> </ul>	<ul style="list-style-type: none"> <li>• authentication: successful</li> <li>• authenticated user: ANONYMOUS</li> </ul>

If Ranger is used for authorization, the authenticated user's identity is used to determine what operations the client is authorized to carry out. As a result, you must ensure that policies in Ranger are set up accordingly.



Cloudera does not recommend that you set this property to requested. It is only useful in a limited number of scenarios and provides a false sense of security. Clients that present no certificates or present an invalid certificate will still be able to establish a connection, but will authenticate as ANONYMOUS. Depending on how your cluster is configured, the ANONYMOUS user might not have access to the required Kafka resources. This can lead to client failure.

### Before you begin

Configure TLS/SSL encryption for the Kafka brokers. For more information, see [Configure TLS/SSL encryption for Kafka brokers](#).

### Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to Configuration.
3. Find and configure the SSL Client Authentication property based on your cluster and requirements.

Cloudera Manager Property	Description
SSL Client Authentication ssl.client.auth	Client authentication mode for SSL connections. This configuration has three valid values, required, requested, and none. If set to required, client authentication is required. If set to requested, client authentication is requested and clients without certificates can still connect. If set to none, which is the default value, no client authentication is required



**Note:** If you are using the SASL\_SSL protocol, for example you authenticate with Kerberos and encrypt with TLS/SSL, and set SSL Client Authentication to required, Kafka will ignore the SSL Client Authentication property. In a case like this, SASL authentication takes precedence. Authenticating the client twice with both SASL and SSL would be redundant.

4. Configure principal mapping rules:
  - a) Find the Kafka Broker Advanced Configuration Snippet (Safety Valve) for kafka.properties property.
  - b) Add principal mapping rules.

For example:

```
ssl.principal.mapping.rules=RULE:^(.*[Cc][Nn])=([a-zA-Z0-9.]*).*$/L,DEFAULT
```



**Note:** For more information on principal mapping rules, see [Principal name mapping](#).

5. Click Save Changes.
6. Restart the Kafka service.

### Results

Kafka brokers are configured for TLS/SSL authentication.

### What to do next

Configure your clients for TLS/SSL authentication.

### Related Information

[Configure TLS/SSL encryption for Kafka brokers](#)

[Principal name mapping](#)

## Configure TLS/SSL authentication for Kafka clients

Kafka supports TLS/SSL authentication (two-way authentication). Client configuration is done by setting the relevant security-related properties for the client.

## About this task

The following steps demonstrate configuration for the console consumer or producer. If you are configuring a custom developed client, see [Java client security examples](#) or [.Net client security examples](#) for code examples.

## Before you begin

- Generate or acquire a key and truststore for your clients which contain all necessary keys and certificates.
- Note down the locations and passwords for the key and truststores. You will need to provide these during configuration.
- If the Certificate Authority of the client certificates is different from the Certificate Authority of the broker certificates, ensure the client Certificate Authority certificate is added to the truststore of the Kafka brokers.

Otherwise, the broker will not trust the certificates used by its clients and a trusted connection will not be established.

## Procedure

1. Create a .properties file.

In this example the file is named client.properties.

2. Add the mandatory properties to the file.

The following configuration example contains all mandatory properties:

```
security.protocol=SSL
ssl.truststore.location=[***PATH TO CLIENT TRUSTSTORE***]
ssl.truststore.password=[***PASSWORD***]
ssl.keystore.location=[***PATH TO CLIENT KEYSTORE***]
ssl.keystore.password=[***PASSWORD***]
ssl.key.password=[***PASSWORD***]
```

3. (Optional) Add additional properties.

Depending on your requirements and broker configuration, other configuration properties might also be needed. The following are some of the most commonly used optional properties:

- ssl.provider
- ss.cipher.suites
- ssl.enabled.protocols
- ssl.truststore.type
- ssl.keystore.type

4. Run the client.

### Console Consumer

```
kafka-console-producer --bootstrap-server [***HOST1:PORT1***] --t
opic [***TOPIC***] --producer.config client.properties
```

### Console Producer

```
kafka-console-consumer --bootstrap-server [***HOST1:PORT1***] --t
opic [***TOPIC***] --consumer.config client.properties
```

## Results

Kafka clients are configured for TLS/SSL authentication.

## Related Information

[Java client security examples](#)

[.Net client security examples](#)

## Principal name mapping

Kafka can be configured to translate certificate subject names into short names. This is done by adding mapping rules to Kafka's configuration. These short names can be used as the unique identifier of the user. Compared to subject names, short names are much easier to manage.

When a client authenticates using a TLS/SSL keystore, by default Kafka assumes that the username for that client is the certificate's subject name, which is usually a Distinguished Name such as the following:

```
cn=alice,cn=groups,cn=accounts,dc=hadoopsecurity,dc=local
```

Working with these long names is difficult. Security policies and group mappings are usually defined in terms of the user's short name (alice) rather than the full Distinguished Name. Kafka can be configured to translate the certificate's subject into a short name that can be used as the unique identifier of the user.

This can be done by adding the necessary mapping rules to the `ssl.principal.mapping.rules` Kafka property. However, this property is not directly configurable in Cloudera Manager. As a result, you need to use the Kafka Broker Advanced Configuration Snippet (Safety Valve) for `kafka.properties` property to add it to your configuration.



**Note:** The `ssl.principal.mapping.rules` property is only supported in Kafka 2.4.0 or higher. In older versions of Kafka, a custom principal builder needs to be created and provided.

The rule takes the form of a regular expression to match the subject name of the certificate and the transformation to apply to the match. The property accepts multiple rules. Each rule has to be separated by a comma. The last rule is usually the `DEFAULT` rule, which uses the full subject name.

For example, consider the following setting:

```
ssl.principal.mapping.rules=RULE:^(^.*[Cc][Nn]=([a-zA-Z0-9.]*).*$/L,DEFAULT
```

This configuration has two rules which are processed in the following order:

1. `RULE:^(^.*[Cc][Nn]=([a-zA-Z0-9.]*).*$/L`
2. `DEFAULT`

The first rule to match the certificate's subject name is used, later ones are ignored. The `DEFAULT` rule is a "catch all" rule. It always matches and does not do any replacement if none of the previous ones were matched.

The regular expression of the first rule, `^(^.*[Cc][Nn]=([a-zA-Z0-9.]*).*$/L`, matches any subject that starts with `CN=`, `cn=`, `Cn=`, or `cN=`, followed by the user's short name, that contains characters ranging between `a-z`, `A-Z`, and `0-9`, followed by any string. It then replaces the matched string with the user's short name. The short name is the content matched inside the parenthesis and is referenced in the second part of the rule as `$1`. The `L` at the end of the rule converts the resulting string to lowercase.

For more information and examples on principal mapping rules, see the Apache Kafka documentation.

### Related Information

[Apache Kafka documentation](#)

## Enable Kerberos authentication for Kafka

Learn how to enable Kerberos Authentication for Kafka.

### About this task

Apache Kafka supports Kerberos authentication, but it is supported only for the new Kafka Producer and Consumer APIs.

## Before you begin

If you already have a Kerberos server, you can add Kafka to your current configuration. If you do not have a Kerberos server, install it before proceeding.

If you already have configured the mapping from Kerberos principals to short names using the `hadoop.security.auth_to_local` HDFS configuration property, configure the same rules for Kafka by adding the `sasl.kerberos.principal.to.local.rules` property to the Advanced Configuration Snippet for Kafka Broker Advanced Configuration Snippet using Cloudera Manager. Specify the rules as a comma separated list.

## Procedure

1. In Cloudera Manager, navigate to Kafka Configuration.
2. Set SSL Client Authentication to none.
3. Set Inter Broker Protocol to SASL\_PLAINTEXT.
4. Click Save Changes.
5. Restart the Kafka service, select Action Restart.
6. Make sure that `listeners = SASL_PLAINTEXT` is present in the Kafka broker logs, by default in `/var/log/kafka/server.log`.
7. Create a `jaas.conf` file with either cached credentials or keytabs.

- To use cached Kerberos credentials, where you use `kinit` first, use this configuration:

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

- If you use a keytab, use this configuration:

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

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

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

9. Test with the Kafka console producer and consumer.

To obtain a Kerberos ticket-granting ticket (TGT):

```
kinit user
```

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

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

11. Verify that your topic exists.

```
kafka-topics --list --bootstrap-server anybroker:9092 --command-config client.properties
```

**12. Run a Kafka console producer.**

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

**13. Run a Kafka console consumer.**

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

## Delegation token based authentication

An overview on Kafka delegation tokens.

Delegation tokens are lightweight authentication method to complement existing SASL authentication. Kafka is designed to support a high number of clients. However, using Kerberos authentication might be difficult in some environments due to the following reasons:

- With Kerberos authentication, all clients need access to a keytab or a TGT. Securely distributing the keytabs requires a lot of effort and careful administration. When the TGT is compromised, it has a high blast radius, especially when the same keytabs are used to access multiple services.
- With Kerberos, client authentication is centralized, and the high number of clients can put a high load on the KDC (Key Distribution Center), resulting in a bottleneck.

Many Hadoop components use delegation tokens to mitigate these problems:

- Delegation tokens allow these components to secure distributed workloads with low administrative overhead.
- It is not required to distribute a Kerberos TGT or keytab, which, if compromised, may grant access to all services.
- A delegation token is strictly tied to its associated service causing less damage if exposed.
- Delegation tokens make credential renewal more lightweight. This is because the renewal is designed in such a way that only the renewer and the service are involved in the renewal process. The token itself remains the same, so parties already using the token do not have to be updated.

### Kafka Delegation Token Basics

Kafka delegation tokens are modeled after Hadoop delegation tokens, and many of their mechanism are the same or very similar. However, this does not mean that they are interchangeable.

Delegation tokens are generated and verified following the HMAC mechanism. There are two basic parts of information in a delegation token, the tokenID (public part) and the HMAC value (private part).

The following list of steps give a generic overview on how delegation tokens are used:

1. The user initially authenticates with the Kafka cluster via SASL, and obtains a delegation token using either the AdminClient APIs or the kafka-delegation-token tool. The principal that created the delegation token is its owner.
2. The delegation token details are securely passed to Kafka clients. This can be achieved by sending the token data over an SSL/TLS encrypted connection or writing them to a secure shared storage.
3. Instead of using Kerberos, the Kafka client uses the delegation tokens for subsequent authentication with the brokers.
4. The token is valid for a certain time period, but it can be:

#### Renewed

A delegation token can be renewed multiple times up until its maximum life before it expires. The token can be renewed by the owner or any other principals the owner sets as “renewer” at time of creation.

#### Revoked

A delegation token can be revoked ahead of its expiry time.

## Broker Configuration

Certain delegation token properties can be configured on a service level in Cloudera Manager. You can enable or disable delegation tokens, as well as configure expiry time and maximum lifetime. For more information, see [Kafka Properties in Cloudera Runtime](#).

## Related Information

[HMAC Wikipedia Page](#)

[Client authentication using delegation tokens](#)

[Kafka Properties in Cloudera Runtime](#)

## Enable or disable authentication with delegation tokens

Learn how to enable or disable Kafka delegation tokens.

### About this task

Delegation token based authentication requires that both the Enable Kerberos Authentication and Enable Delegation Tokens properties are set to true. The default value of the Enable Delegation Tokens property is true, but will have no effect until Kerberos is also enabled.

Although enabling delegation tokens enables authentication between clients and servers using the SASL/SCRAM mechanism, it is only as a vehicle for delegation tokens. Using SCRAM credentials is not supported otherwise.

Sensitive delegation token metadata is stored in Zookeeper. It is recommended to restrict access on Zookeeper nodes to prevent access to sensitive delegation token related data through Zookeeper. The connection between Kafka and Zookeeper is not encrypted, therefore, it is also recommended to use delegation tokens only if no unauthorized person can read and manipulate the traffic between these services.

Delegation tokens can be enabled separately for each Kafka service.

### Before you begin

A secure Kafka cluster with Kerberos authentication enabled is required for delegation tokens to function.

### Procedure

1. In Cloudera Manager select the Kafka service.
2. Select Configuration and find the Enable Delegation Tokens property.
3. Enable or disable delegation tokens for all required services by checking or unchecking the checkbox next to the name of the service.
4. Click Save Changes.
5. Perform a Rolling Restart:
  - a) Return to the Home page by clicking the Cloudera Manager logo.
  - b) Go to the Kafka service and select ActionsRolling Restart.
  - c) Check the Restart roles with stale configurations only checkbox and click Rolling restart.
  - d) Click Close when the restart has finished.

### Results

Delegation tokens are enabled or disabled for the selected Kafka services. If delegation tokens were enabled, then the necessary secrets and settings are generated.

## Related Information

[Enable Kerberos authentication for Kafka](#)

[Kafka security hardening with Zookeeper ACLs](#)

## Manage individual delegation tokens

A list of example actions you can issue to manage individual delegation tokens.

The functionality that's needed to manage and use delegation tokens is accessible using the AdminClient APIs or the kafka-delegation-tokens tool. All of their operations are allowed only via SASL authenticated channels.

Both the API and the script provide the following actions:



**Note:** The examples presented show how these actions can be executed with the kafka-delegation-tokens tool.

### Issue, and store for verification

The owner of the token is the currently authenticated principal. A renewer can be specified when requesting the token.

```
kafka-delegation-tokens --bootstrap-server hostname:port --create
--max-life-time-period -1 --command-config client.properties
--renewer-principal User:user1
```

### Renew

Only the owner and the principals that are renewers of the delegation token can extend its validity by renewing it before it expires. A successful renewal extends the Delegation Token's expiration time for another renew-interval, until it reaches its max lifetime. Expired delegation tokens cannot be used to authenticate, the brokers will remove expired delegation tokens from the broker's cache and from Zookeeper.

```
kafka-delegation-tokens --bootstrap-server hostname:port --renew
--renew-time-period -1 --command-config client.properties --hmac
1AYYSFmLs4bTjf+1TZ1LCHR/ZZFNA==
```

### Remove

Delegation tokens are removed when they are canceled by the client or when they expire.

```
kafka-delegation-tokens --bootstrap-server hostname:port --expire
--expiry-time-period -1 --command-config client.properties
--hmac 1AYYSFmLs4bTjf+1TZ1LCHR/ZZFNA==
```

### Describe

Tokens can be described by owners, renewers or the Kafka super user.

```
kafka-delegation-tokens --bootstrap-server hostname:port --describe
--command-config client.properties --owner-principal User:user1
```



**Note:** In Apache Kafka, principals that have the describe permission on the token resource can also describe the token.

## Rotate the master key/secret

Learn how to rotate the delegation token Master Key/Secret.

### About this task

The brokers generate and verify delegation tokens using a secret called delegation.token.master.key. This secret is generated by Cloudera Manager and securely passed to Kafka brokers when authentication with delegation tokens is enabled. You can change the secret with the Cloudera Manager API. This should be done if the secret becomes compromised, or simply as a precautionary measure.



**Important:** Clients that were already connected to brokers before starting the process, will continue to work even after the master key/secret is rotated. However, any new connections (authentication requests), as well as renew and expire requests with old tokens can fail.

## Procedure

1. Expire existing tokens.

```
kafka-delegation-tokens --bootstrap-server hostname:port --expire --expiry-time-period -1 --command-config client.properties --hmac LAYYSFmLs4bTjf+1TZ1LCHR/ZZFNA==
```

2. Generate a new master key

```
curl -X PUT -u "user" -H "content-type:application/json" -i "https://cloudera manager host:7183/api/v31/clusters/cluster name/service s/kafka service name/config" -d '{"items" : [ {"name" : "delegation.token.master.key", "value" : "'$(openssl rand -base64 24)'" , "sensitive" : true}]}'
```

3. Perform a Rolling Restart:

- a) In Cloudera Manager go to the Kafka service and select ActionsRolling Restart.
- b) Check the Restart roles with stale configurations only checkbox and click Rolling restart.
- c) Click Close when the restart has finished.

4. Reauthenticate with all clients.

This generates the new tokens.

## Results

The Master Key/Secret is rotated.

## Client authentication using delegation tokens

An overview of the methods available for configuring clients to use delegation tokens.

Brokers authenticate clients by verifying the delegation tokens provided by the client against the stored delegation tokens. Delegation token authentication makes use of SASL/SCRAM authentication mechanism under the hood. You can configure Kafka clients in two ways, to use individually assigned delegation tokens or to use a common delegation token.

### Configure clients on a producer or consumer level

Learn how to configure client authentication using delegation tokens on a producer or consumer level.

### About this task

You can set up client authentication by configuring the JAAS configuration property for each client. The JAAS configuration property can be set in the producer.properties or consumer.properties file of the client. With this configuration method, you have the ability to specify different token details for each Kafka client within a JVM. As a result you can configure Kafka clients in a way that each of them use a unique token for authentication.

## Procedure

Configure the JAAS configuration property for each client.

Example Configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required \
  username="tokenID" \
  password="LAYYSFmLs4bTjf+1TZ1LCHR/ZZFNA==" \
  tokenauth="true";
```

There are three options that need to be specified. These are the username, password and tokenauth options.



The username and password options specify the token ID and token HMAC. The tokenauth option expresses the intent to use token authentication to the server.

### Configure clients on an application level

Learn how to configure client authentication using delegation tokens on an application level.

#### About this task

With this configuration method, you can set up all clients within a JVM to use the same delegation token for authentication.

#### Procedure

1. Add a KafkaClient entry with a login module item to your JAAS configuration file.

The module has to specify the username, password and tokenauth options.

Example Configuration:

```
KafkaClient {  
  org.apache.kafka.common.security.scram.ScramLoginModule required  
    username="tokenID"  
    password="1AYYSFmLs4bTjf+1TZ1LCHR/ZZFNA=="  
    tokenauth="true";  
}
```

The username and password options specify the token ID and token HMAC. The tokenauth option expresses the intent to use token authentication to the server.

2. Pass the location of your JAAS configuration file as a JVM parameter through a command line interface.

This sets the JAAS configuration on the Java process level.

```
export KAFKA_OPTS="-Djava.security.auth.login.config=[PATH_TO_JAAS.CONF]"
```

## Kafka security hardening with Zookeeper ACLs

Learn how to restrict or unlock access to Kafka Metadata in Zookeeper.

### Restrict access to Kafka metadata in Zookeeper

Learn how to restrict access to Kafka metadata in Zookeeper.

#### About this task

Locking down znodes in Zookeeper can be used to protect Kafka metadata against unauthorized access. Direct manipulation of metadata in Zookeeper is not only dangerous for the health of the cluster, but can also serve as an entry point for malicious users to gain elevated access who can then alter the owner or renewer of delegation tokens. Access to Kafka metadata in Zookeeper is restricted by default. Follow these steps if you have previously unlocked access, but want to re-enable access restrictions.

#### Before you begin

A secure Kafka cluster with Kerberos authentication enabled is required.

## Procedure

### 1. Enable the use of secure ACLs:

This can be achieved by setting the `zookeeper.set.acl` configuration parameter to `true`.

- In Cloudera Manager select the Kafka service.
- Select Configuration.
- Find the Enable Zookeeper ACL property.
- Set the property to `true` by checking the checkbox next to the name of the role group.

### 2. Perform a Rolling Restart:

- Return to the Home page by clicking the Cloudera Manager logo.
- Check the Restart roles with stale configurations only checkbox and click Rolling restart.
- Click Close when the restart has finished.

### 3. Pass the JAAS configuration file location as a JVM parameter through a command line interface.

You can do this by setting the value of the `KAFKA_OPTS` environment variable to `-Djava.security.auth.login.config=[PATH_TO_JAAS.CONF]`.

```
export KAFKA_OPTS="-Djava.security.auth.login.config=[PATH_TO_JAAS.CONF]"
```

Replace `[PATH_TO_JAAS.CONF]` with the path to your JAAS configuration file.

### 4. Run the `zookeeper-security-migration` tool with the `zookeeper.acl` option set to `secure`.

```
zookeeper-security-migration --zookeeper.connect [HOSTNAME]:[PORT]/[ZOOKEEPER_KAFKA_CHROOT] --zookeeper.acl secure
```

Replace `[ZOOKEEPER_KAFKA_CHROOT]` with the value of the `zookeeper.chroot` property. To view the value of the property, in Cloudera Manager go to **Kafka Configuration** and search for `zookeeper.chroot`.

The tool traverses the corresponding sub-trees changing the ACLs of the `znodes`.

### 5. Reset the ACLs on the root node to allow full access:

Resetting the ACLS on the root node is required because the `zookeeper-security-migration` tool also changes the ACLs on the root `znode`. This leads to the failure of the Zookeeper canary tests, which subsequently makes the service display as unhealthy in Cloudera Manager.



**Important:** This step is only necessary if the `zookeeper.chroot` parameter of the broker is set to `/`.



**Note:** Because the Kafka metadata at this point is already restricted, only authorized users or Zookeeper super users can complete this step.

- Change the `JVMFLAGS` environment variable to `-Djava.security.auth.login.config=[PATH_TO_JAAS.CONF]`.

```
export JVMFLAGS="-Djava.security.auth.login.config=[PATH_TO_JAAS.CONF]"
```

- Start the `zookeeper` client.

```
zookeeper-client -server $(hostname -f):2181
```

- Enter the following to reset the ACLs of the root node.

```
setAcl / world:anyone:crdwa
```

## Results

Kafka metadata in Zookeeper is restricted via ACLS. Administrative operations, for example topic creation, deletion, any configuration changes and so on, can only be performed by authorized users.

## Related Information

[Enable Kerberos authentication for Kafka](#)

## Unlock Kafka metadata in Zookeeper

Learn how to unlock access to Kafka metadata in Zookeeper

## Before you begin

A secure Kafka cluster with Kerberos authentication enabled is required.

## Procedure

### 1. Disable the use of secure ACLs:

This can be achieved by setting the `zookeeper.set.acl` configuration parameter to false.

- a) In Cloudera Manager select the Kafka service.
- b) Select Configuration.
- c) Find the Enable Zookeeper ACL property.
- d) Set the property to false by unchecking the checkbox next to the name of the role group.

### 2. Perform a Rolling Restart:

- a) Return to the Home page by clicking the Cloudera Manager logo.
- b) Go to the Kafka service and select Actions Rolling Restart.
- c) Check the Restart roles with stale configurations only checkbox and click Rolling restart.
- d) Click Close when the restart has finished.

### 3. Run the `zookeeper-security-migration` tool with the `zookeeper.acl` option set to unsecure

```
zookeeper-security-migration --zookeeper.connect [HOSTNAME]:[PORT]/[ZOOKEEPER_KAFKA_CHROOT] --zookeeper.acl unsecure
```

Replace `[ZOOKEEPER_KAFKA_CHROOT]` with the value of the `zookeeper.chroot` property. To view the value of the property, in Cloudera Manager go to `Kafka Configuration` and search for `zookeeper.chroot`.

## Results

The tool traverses the corresponding sub-trees changing the ACLs of the znodes. Access to Kafka metadata stored in Zookeeper becomes unrestricted.

## Related Information

[Enable Kerberos authentication for Kafka](#)

## LDAP authentication

You can configure Kafka to make use of LDAP credentials for client to broker authentication. In order to enable LDAP authentication you must configure both brokers and clients accordingly.

## Configure LDAP authentication for Kafka brokers

Learn how to configure LDAP authentication for Kafka brokers.

## About this task

You can enable Kafka to use LDAP credentials for client to broker authentication. Broker configuration is done by configuring the required properties in Cloudera Manager.

## Before you begin



**Important:** Kafka Brokers provisioned after July 28, 2022 use LDAP authentication by default. Previously PAM was the default. If your cluster was provisioned following this date, LDAP is already configured and you do not need to complete the following steps. If you are transitioning from PAM to LDAP, you might need to reconfigure your clients and ensure that your clients use short names. Specifying usernames in the client configuration that contain the realm/domain are not supported.

## Procedure

1. In Cloudera Manager, select the Kafka service.
2. Select Configuration.
3. Enable LDAP authentication:
  - a) Find the SASL/PLAIN Authentication property.
  - b) Click the radio button next to LDAP. Do this for all required Kafka services.
4. Click Save Changes.
5. Restart the Kafka service.

## Results

LDAP authentication is configured for the brokers.

## What to do next

Configure clients to use LDAP authentication.

## Related Information

[Configure Kafka LDAP authentication for Kafka clients](#)

# Configure Kafka LDAP authentication for Kafka clients

Learn how to configure Kafka clients for LDAP authentication.

## About this task

You can enable Kafka to use LDAP credentials for client to broker authentication. Client configuration is done by adding the required properties to the client's `client.properties` file.

## Before you begin

- Clients connecting to Data Hub provisioned clusters require a CDP user account that provides access to the required CDP resources. Verify that a CDP user account with the required roles and permissions is available for use. If not, create one. Any type of CDP user account can be used. If you are creating a new account to be used by Kafka clients, Cloudera recommends that you create a machine user account. For more information, see [User Management](#) in the Cloudera Management Console documentation.
- In addition to the CDP user account having access to the required CDP resources, the user account also needs to have the correct policies assigned to it in Ranger. Otherwise, the client cannot perform tasks on Kafka resources. These policies are specified within the Ranger instance that provides authorization to the Kafka service you want to connect to. For more information, see the [Cloudera Runtime documentation on Apache Ranger](#) and the [Kafka specific Ranger documentation](#).
- If you are transitioning from PAM to LDAP, ensure that the usernames you specify in the client configuration are short names. Usernames that include the realm/domain are not supported.

## Procedure

1. Set the SASL mechanism to PLAIN.

Add the following property to the client.properties file.

```
sasl.mechanism=PLAIN
```

2. Configure the security protocol.

You can either use SASL\_SSL or SASL\_PLAINTEXT. Which security protocol you use will depend on whether or not SSL encryption is enabled on the broker. Add one of the following properties to the client.properties file.

- If encryption is enabled, use SASL\_SSL:

```
security.protocol=SASL_SSL
```

- If encryption is not enabled, use SASL\_PLAINTEXT:

```
security.protocol=SASL_PLAINTEXT
```



**Note:** In a public network Cloudera recommends that you use SASL\_SSL as LDAP credentials can become exposed.

3. Configure the JAAS.

You have two options when configuring the JAAS:

- a) Embed the required properties in the client.properties file with the sasl.jaas.config property.

```
sasl.jaas.config= \
org.apache.kafka.common.security.plain.PlainLoginModule required \
  username=["***USERNAME***"] \
  password=["***PASSWORD***"] ;
```

Replace `***USERNAME***` and `***PASSWORD***` with the credentials of the CDP user you set up for the client in Management Console. For the username, ensure that you use short names. Specifying usernames that contain the realm/domain are not supported.

- b) Use a separate JAAS config file:

1. Add a KafkaClient entry with a login module item to your JAAS configuration file.

Example configuration:

```
KafkaClient {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username=["***USERNAME***"]
  password=["***PASSWORD***"] ;
};
```

Replace `***USERNAME***` and `***PASSWORD***` with the credentials of the CDP user you set up for the client in Management Console. For the username, ensure that you use short names. Specifying usernames that contain the realm/domain are not supported.

2. Pass the location of your JAAS configuration file as a JVM parameter through a command line interface

```
export KAFKA_OPTS="-Djava.security.auth.login.config=[***PATH TO
JAAS.CONF***]"
```

Replace `***PATH TO JAAS.CONF***` with the location of the JAAS configuration file you created.

## Results

LDAP authentication is configured for the client.

## PAM authentication

You can configure Kafka to use the PAM authentication mechanism for client to broker authentication. In order to enable PAM authentication you must configure both brokers and clients accordingly.

### Configure PAM authentication for Kafka brokers

Learn how to configure PAM authentication for Kafka brokers.

#### About this task

You can enable Kafka to use PAM for client to broker authentication. Broker configuration is done by configuring the required properties in Cloudera Manager.

#### Procedure

1. In Cloudera Manager select the Kafka service.
2. Select Configuration.
3. Enable PAM authentication:
  - a) Find the SASL/PLAIN Authentication property.
  - b) Click the radio button next to PAM. Do this for all required Kafka services.
4. Configure the PAM service name:
  - a) Find the PAM Service property.
  - b) Enter a valid PAM service name. The property defaults to login.
5. Click Save Changes.
6. Restart the Kafka service

#### Results

PAM authentication is configured for the brokers.

#### What to do next

Configure clients to use PAM authentication.

### Configure PAM authentication for Kafka clients

Learn how to configure Kafka clients for PAM authentication.

#### About this task

You can enable Kafka to use PAM for client to broker authentication. Client configuration is done by adding the required properties to the client's client.properties file.

#### Procedure

1. Set the SASL mechanism to PLAIN.  
Add the following property to the client.properties file.

```
sasl.mechanism=PLAIN
```

## 2. Configure the security protocol.

You can either use SASL\_SSL or SASL\_PLAINTEXT. Which security protocol you use will depend on whether or not SSL encryption is enabled on the broker. Add one of the following properties to the client.properties file.

- If encryption is enabled, use SASL\_SSL:

```
security.protocol=SASL_SSL
```

- If encryption is not enabled, use SASL\_PLAINTEXT:

```
security.protocol=SASL_PLAINTEXT
```



**Note:** In a public network Cloudera recommends that you use SASL\_SSL as credentials can become exposed.

## 3. Configure the JAAS.

You have two options when configuring the JAAS:

- a) Embed the required properties in the client.properties file with the sasl.jaas.config property.

```
sasl.jaas.config= \
org.apache.kafka.common.security.plain.PlainLoginModule required \
  username="[USERNAME]" \
  password="[PASSWORD]";
```

Replace *[USERNAME]* and *[PASSWORD]* with a valid username and password.

- b) Use a separate JAAS config file:

1. Add a KafkaClient entry with a login module item to your JAAS configuration file.

Example configuration:

```
KafkaClient {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="[USERNAME]"
  password="[PASSWORD]";
};
```

Replace *[USERNAME]* and *[PASSWORD]* with a valid username and password.

2. Pass the location of your JAAS configuration file as a JVM parameter through a command line interface

```
export KAFKA_OPTS="-Djava.security.auth.login.config=[PATH_TO_JAAS.CONF]"
```

Replace *[PATH\_TO\_JAAS.CONF]* with the location of the JAAS configuration file you created.

### Results

PAM authentication is configured for the client.

## Authorization

### Ranger

Learn more about authorization with Ranger.

You can use Ranger to provide authorization for Kafka. For more information, see [Using Ranger to Provide Authorization in CDP](#).

### Related Information

[Using Ranger to Provide Authorization in CDP](#)

## Enable authorization in Kafka with Ranger

Learn how to enable Ranger authorization for Kafka.

### Before you begin

Ranger authorization requires that at least one of the following authentication mechanisms is enabled in Kafka:

- Kerberos
- Two-way TLS/SSL
- LDAP
- PAM

### Procedure

1. In Cloudera Manager select the Kafka service.
2. Select Configuration and find the RANGER Service property.
3. Check the checkbox next to the name of the Ranger service that you want this Kafka service to depend on.
4. Click Save Changes.
5. Restart the Kafka service.

### Results

Ranger authorization for Kafka is enabled. The Kafka service depends on the selected Ranger service for authorization.

### What to do next

Configure resource-based services and policies for Kafka. Additionally, configure which resource-based service should be used for authorization.

### Related Information

[Configure a resource-based service: Kafka](#)

[Configure a resource-based policy: Kafka](#)

[Configure the resource-based Ranger service used for authorization](#)

[Enable Kerberos authentication for Kafka](#)

[Channel encryption](#)

[Configure TLS/SSL client authentication for Kafka brokers](#)

[LDAP authentication](#)

[PAM authentication](#)

## Configure the resource-based Ranger service used for authorization

Learn how to configure the resource-based Ranger service used by Kafka for authorization.

### About this task

You can configure which resource-based service in Ranger is used by Kafka for authorization. This is controlled by the Ranger service name for this Kafka cluster property which can be set in Cloudera Manager. The property can be configured in two ways:

#### Set a user defined string



The property accepts a user defined string. Alphanumeric characters as well as underscores are allowed, for example, `kafka_1`. In this case, Kafka checks if a resource-based service in Ranger matching the defined string exists. If it does, Kafka will use that resource-based service. If it does not, Kafka automatically creates and uses a new resource-based service. The name of the newly created resource-based service will be the name that was defined in the property.

#### Set `{{GENERATED_RANGER_SERVICE_NAME}}`

The property accepts `{{GENERATED_RANGER_SERVICE_NAME}}` as a valid value. When this specific placeholder value is set, Cloudera Manager generates a name for a new resource-based service. The generated name is based on the Kafka service name and the cluster name. Kafka then checks whether a resource-based service with the generated name exists in Ranger. If it does, Kafka will use that resource-based service. If it does not, Kafka automatically creates and uses a new resource-based service. The name of the newly created resource-based service will be the generated name.

In both scenarios, the creation of the role-based service happens once the Kafka service is restarted.

The property by default is set to `cm_kafka`. However, for clusters provisioned with the Streams Messaging cluster definitions in CDP Data Hub, the default is `{{GENERATED_RANGER_SERVICE_NAME}}`.

Configuring this property can prove useful when you have multiple Kafka clusters that use the same Ranger service for authorization, but you want to define unique Ranger policies for each Kafka cluster.

### Procedure

1. In Cloudera Manager select the Kafka service.
2. Select Configuration and find the Ranger service name for this Kafka cluster property.
3. Configure the property.  
You have two choices:
  - Set the property to a user defined string.
  - Set the property to `{{GENERATED_RANGER_SERVICE_NAME}}`.
4. Click Save Changes.
5. Restart the Kafka service.

### Results

The selected resource-based service is configured and is used by Kafka for authorization. If the resource-based service does not exist, it will be automatically created in Ranger.

### Related Information

[Apache Ranger Authorization](#)

## Governance

You can monitor and govern Kafka metadata in Atlas. This can be done by enabling and configuring the Atlas hook in Kafka.

### Configuring the Atlas hook in Kafka

An Atlas hook collects metadata from Kafka and transfers it to Atlas so that you can manage, govern, and monitor Kafka metadata and metadata lineage in the Atlas UI. You can enable and configure an Atlas hook in Kafka.

## About this task

The following instructions walk you through how you enable the Atlas hook, and how you configure the topic and client metadata namespaces used by the hook. The namespaces you configure for the hook in Kafka, are used in Atlas, to group and represent Kafka entities. Once the hook is set up and configured, Kafka metadata will be available in Atlas.

Enabling and configuring the hook only imports and exposes newly created Kafka topics. Existing topics are not imported automatically. To access existing topics, you must use the `kafka-import.sh` tool and import them manually into Atlas.

## Before you begin

- Ensure that an Atlas service is deployed on the Kafka cluster or the data context cluster.
- Optional: Import existing Kafka topics into Atlas with the `import-kafka.sh` tool.

## Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to Configuration.
3. Find and enable the Enable Auditing to Atlas property.

This property enables or disables the Atlas hook in Kafka.

4. Specify the topic and client metadata namespaces.

The topic and client metadata namespaces are configured with the Atlas metadata namespace for Kafka Topics and Atlas metadata namespace for Kafka Clients properties. How you configure these properties depends on your environment and use case. Cloudera recommends that you follow these guidelines when configuring namespaces:

- Use the same topic namespace you used with the `kafka-topics.sh` tool if you manually imported topics.
- Use identical client and topics namespaces if only a single Kafka cluster is audited by Atlas.

In this case, you can also consider using the default namespace, which is `cm`.

- Use unique topic namespaces in environments where there are multiple Kafka clusters audited by Atlas.

This is a recommended practice to avoid the conflict of topic entities.

- Configure client namespaces based on your use case.

Unlike topic namespaces, client namespaces do not have to be unique even if there are multiple Kafka clusters in your environment. For example, if there is an application communicating with multiple Kafka clusters, and it is using the same `client.id`, the client metadata namespace can be set to the same value for all Kafka clusters.

This way, the application is represented as a single producer or consumer entity in Atlas.

5. Click Save Changes.
6. Restart the Kafka service.

## Results

The Atlas hook in Kafka is enabled and configured. Kafka topic, consumer, producer, and consumer group metadata can be monitored using the Atlas UI.

## What to do next

Learn more about the Atlas hook as well as Kafka metadata collection. For more information see *Kafka metadata collection*.

## Related Information

[Kafka metadata collection](#)

## Inter-broker security

Kafka can expose multiple communication endpoints, each supporting a different protocol. Supporting multiple communication endpoints enables you to use different communication protocols for client-to-broker communications and inter-broker communications.

The security protocol used for inter-broker communication is controlled by the Inter Broker Protocol Cloudera Manager property. By default this property is set to `INFERRED`, which sets the security protocol based on how other security properties of the broker are configured.

The `INFERRED` setting configures the protocol according to the following logic:

Kerberos or LDAP enabled	TLS/SSL enabled	Protocol
YES	YES	SASL_SSL
YES	NO	SASL_PLAINTEXT
NO	YES	SSL
NO	NO	PLAINTEXT

Cloudera recommends that you use the protocol set by `INFERRED`. However, you can change this setting and use a specific protocol. This can be done by setting the Inter Broker Protocol property to the protocol that you want to use for inter-broker communication.

Changing the inter-broker protocol from the default is primarily useful for the following reasons:

### Improving performance

Enabling TLS/SSL is known to have performance overhead. If your Kafka brokers are behind a firewall and are not susceptible to network snooping, you can consider enabling TLS/SSL for client-to-broker communication, but keep inter-broker communication set as `PLAINTEXT`. A configuration like this can result in a better performing Kafka cluster.

### Securing a non-secure Kafka deployment without downtime

It is possible to migrate from a non-secure Kafka configuration to a secure Kafka configuration without downtime. This can be achieved with a rolling restart and by setting the inter-broker protocol to a protocol that is supported by all brokers until all brokers are updated to support the new protocol. For example, if you have a Kafka cluster that needs to be configured to enable Kerberos without downtime, you would take the following steps:

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

## Configuring multiple listeners

Kafka brokers can simultaneously listen to connection requests on multiple ports with various protocols. By default Cloudera Manager automatically configures the ports and the protocols used by the brokers to listen to requests. However, manual configuration is possible and may be required in an advanced deployment.

### About this task

Kafka Brokers support listening for connections on multiple ports with different protocols. For example, a broker is capable of listening on port 9092 for `PLAINTEXT` requests and 9093 using TLS/SSL simultaneously. Which ports a broker listens to is controlled by the `listeners` Kafka broker property.

In Cloudera Manager, the listeners property is not available directly for configuration. Instead, it is set by Cloudera Manager automatically based on how other security properties are configured.

For example, if TLS/SSL is enabled, `Enable TLS/SSL for Kafka Broker` is set to true, Cloudera Manager automatically configures the Kafka broker to listen to TLS/SSL requests on port 9093. That is, the listeners property is set to `SSL://[***KAFKA BROKER FQDN***]:9093`.

It is usually sufficient to rely on Cloudera Manager to automatically configure listeners for you. However, in a deployment where the clients use various protocols and ports to establish a connection with the broker, you need to configure the broker so that it listens on all ports and with all protocols used by the clients. This configuration has to be done manually with the help of an advanced configuration snippet.

Complete the following steps to manually configure listeners for Kafka brokers.

### Before you begin

During configuration, ensure that listeners are set individually for each broker, using each broker's FQDN.

### Procedure

1. In Cloudera Manager, select the Kafka service.
2. Go to Instances.
3. Configure listeners using an advanced configuration snippet:

Repeat the following steps for each Kafka broker role.

- a) Click on a Kafka broker role.
- b) Go to Configuration.
- c) Find the Kafka Broker Advanced Configuration Snippet (Safety Valve) for `kafka.properties` property and configure listeners.

For example:

```
listeners=SASL_SSL://[***KAFKA BROKER FQDN***]:9093,SSL://[***KAFKA  
BROKER FQDN***]:9094
```

This example shows a configuration where the broker is accepting both SASL\_SSL and SSL requests.



**Important:** Ensure that you configure a listener for all protocols and ports that you want the broker to listen to.

- d) Click Save Changes.
4. Restart the Kafka service.

### Results

The Kafka broker now listens for connection requests on the configured ports with the configured protocols.