Cloudera Runtime 7.3.2

Securing Apache Kafka

Date published: 2019-08-22
Date modified: 2026-03-31

https://docs.cloudera.com/

Legal Notice

Contents

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 Administration Settings Java 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
ssl.provider	The name of the security provider used for TLS/SSL connections. Default is the default security provider of the JVM.
ssl.cipher.suites	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.
ssl.enabled.protocols	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.

3. 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.

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

Results

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

Configure TLS/SSL for KRaft Controllers

Learn how to configure TLS/SSL for KRaft Controllers.

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.

Table 2: KRaft TLS/SSL configuration properties

Cloudera Manager Property	Description
Enable TLS/SSL for Kraft Controller ssl_enabled	Encrypt communication between clients and KRaft Controller using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).

Cloudera Manager Property	Description
KRaft Controller 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 KRaft Controller is acting as a TLS/SSL server. The keystore must be in the format specified in Administration Settings Java Keystore Type .
KRaft Controller TLS/SSL Server JKS Keystore File Password ssl_server_keystore_password	The password for the KRaft Controller keystore file.
KRaft Controller TLS/SSL Server JKS Keystore Key Password ssl_server_keystore_keypassword	The password that protects the private key contained in the keystore used when KRaft Controller is acting as a TLS/SSL server.
KRaft Controller TLS/SSL Trust Store File ssl_client_truststore_location	The location on disk of the trust store, in .jks format, used to confirm the authenticity of TLS/SSL servers that KRaft Controller 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.
KRaft Controller TLS/SSL Trust Store Password ssl_client_truststore_password	The password for the KRaft Controller 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.
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.

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

What to do next

TLS/SSL encryption is configured for the KRaft Controller role.

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 3: SSL Client Authentication options and outcomes

SSL Client Authentication	Client certificate is presented	Client certificate is not presented
required	• authentication: successful • authenticated user: *[***SUBJECT FROM THE CERTIFICATE***]*	• authentication: fails
requested	• authentication: successful • authenticated user: *[***SUBJECT FROM THE CERTIFICATE***]*	• authentication: successful • authenticated user: ANONYMOUS
none	• authentication: successful • authenticated user: ANONYMOUS	• authentication: successful • authenticated user: ANONYMOUS

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.]*).*$/$1/L,DEF
AULT
```

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.]*).*$/$1/L,DEFAULT
```

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

1. RULE:^[Cc][Nn]=([a-zA-Z0-9.]*).*$/$1/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.]*).*$, 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.l

ocal.rules property to the Advanced Configuration Snippet for Kafka Broker Advanced Configuration Snippet using Cloudera Manager. Specify the rules as a comma separated list.

Important: If your auth_to_local rules include commas (for example, regex quantifiers like {2,3}), the system might interpret those commas as rule separators when propagated to sasl.kerberos.principal.to.local.rules. This interpretation can break Kafka broker startup.

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 c
    lient.properties
    ```
12. Run a Kafka console producer.

    ```
    kafka-console-producer --broker-list ANYBROKER:9092 --topic test1 --produc
    er.config client.properties
    ```

13. Run a Kafka console consumer.

```
kafka-console-consumer --topic test1 --from-beginning --bootstrap-se
rver 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 Cloudera Manager Configuration Properties Reference.

Related Information

HMAC Wikipedia Page

Client authentication using delegation tokens

Cloudera Manager Configuration Properties Reference

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 the metadata service. In ZooKeeper-based deployments, tokens are stored in ZooKeeper. In KRaft mode, tokens are stored in the metadata log (the __cluster_metadata topic). For ZooKeeper-based deployments, it is recommended to restrict access on ZooKeeper nodes to prevent access to sensitive delegation token related data. The connection between Kafka and ZooKeeper is not encrypted, therefore, it is recommended to use delegation tokens only if no unauthorized person can read and manipulate the traffic between these services. For KRaft deployments, ensure that controllers are on a private network or that all communications between brokers and controllers are encrypted.

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  --crea
te   --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 the metadata service.

```
kafka-delegation-tokens --bootstrap-server HOSTNAME:PORT --renew
    --renew-time-period -1 --command-config CLIENT.PROPERTIES --h
mac LAYYSFMLS4BTJF+LTZ1LCHR/ZZFNA==
```

Remove

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

```
kafka-delegation-tokens --bootstrap-server HOSTNAME:PORT --expir
e   --expiry-time-period -1   --command-config CLIENT.PROPERTIES
 --hmac LAYYSFMLS4BTJF+LTZ1LCHR/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-princi
pal 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 --expi
ry-time-period -1 --command-config CLIENT.PROPERTIES --hmac LAYYSFMLS4BTJF
+LTZ1LCHR/ZZFNA==
```

2. Generate a new master key

```
curl -X PUT -u "USER" -H "content-type:application/json" -i "htt
ps://CLOUDERA MANAGER HOST:7183/api/v31/clusters/CLUSTER NAME/service
s/KAFKA SERVICE NAME/config" -d '{"items" : [ {"name" : "delegation.toke
n.master.key","value" : "'$(openssl rand -base64 24)'","sensitive" : tru
e}]}'
```

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 as well as other mandatory properties for each client. All properties 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 each client.

Example Configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule  re
quired \
    username="TOKENID" \
    password="LAYYSFMLS4BTJF+LTZ1LCHR/ZZFNA==" \
```

```
    tokenauth="TRUE";
security.protocol=SASL_SSL
sasl.mechanism=SCRAM-SHA-256
```

Within the JAAS configuration, 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. The configuration is similar to the following example:

   ```
   KafkaClient {
   org.apache.kafka.common.security.scram.ScramLoginModule  required
       username="TOKENID"
       password="LAYYSFMLS4BTJF+LTZ1LCHR/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. Configure the following properties for your clients.

   These properties are added to producer.properties or consumer.properties file that the client uses.

   ```
   security.protocol=SASL_SSL
   sasl.mechanism=SCRAM-SHA-256
   ```
3. 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]"
   ```

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. Go to 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. Configure the allow list of LDAP URLs.

   You can restrict which LDAP URLs Kafka is allowed to connect to by configuring the `com.cloudera.kafka.ldap.allowed.urls` Java option. By specifying a trusted list of LDAP servers, you can have stricter control over the LDAP servers Kafka can access.
   a) Find the Additional Broker Java Options property.
   b) Add the `com.cloudera.kafka.ldap.allowed.urls` Java option. For example:

   ```
   -Dcom.cloudera.kafka.ldap.allowed.urls=HTTP://WWW.LDAP-EXAMPLE-1.COM,HTTP://WWW.LDAP-EXAMPLE-2.COM
   ```
5. Click the Save Changes button.
6. 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

Cloudera Blog – How to configure clients to connect to Apache Kafka Clusters securely – Part 2: LDAP

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 Cloudera Data Hub provisioned clusters require a Cloudera user account that provides access to the required Cloudera resources. Verify that a Cloudera user account with the required roles and permissions is available for use. If not, create one. Any type of Cloudera 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 Cloudera user account having access to the required Cloudera 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 Cloudera 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 Cloudera 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.C
      ONF]"
      ```

      Replace *[PATH_TO_JAAS.CONF]* with the location of the JAAS configuration file you created.

Results

PAM authentication is configured for the client.

OAuth2 authentication

Learn about OAuth2 authentication and how OAuth2 is implemented in Apache Kafka.

OAuth2 is a set of open standards used for access delegation. It is used for enabling applications and services to access other services or resources. Using OAuth2 in Kafka is made possible by an OAuth2-compatible token-based mechanism called SASL OAUTHBEARER. The OAUTHBEARER implementation in Cloudera Kafka uses and accepts signed JSON Web Tokens (JWTs). The mechanism can be used for client to broker authentication.

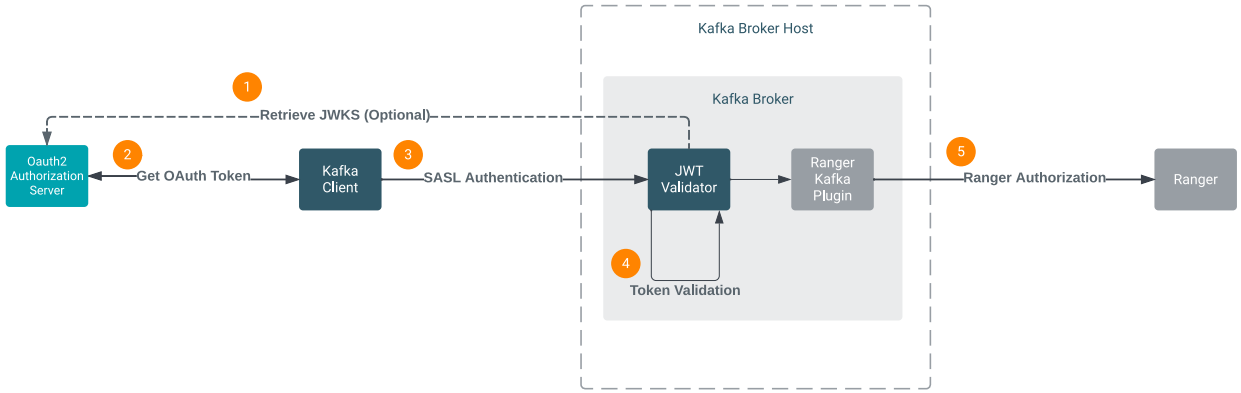
Important: Inter-broker authentication using OAuth2 is not supported.

To ensure that the information transferred through JWTs is trusted, that is, to make JWTs tamper-proof, they can be signed. JWTs can be signed with a secret (using the HMAC algorithm) or with a key pair consisting of a public and private key using RSA or ECDSA. While both signature methods are viable options, Cloudera recommends that you use RSA or ECDSA. This is because using RSA or ECDSA also certifies that the token is issued by the authorization server.

Note: Cloudera Kafka only supports the client_credentials grant type for access tokens.

The authentication flow involves the client, an OAuth2 authorization server, and the Kafka broker.

Figure 1: OAuth2 authentication flow



1. The Kafka Broker retrieves the JSON Web Key Set (JWKS) from the authorization server.

   JWKS retrieval happens at broker startup and is optional. It is only done if the broker is configured for online JWKS access. The broker can also be configured to access a locally available JWKS (not shown on the diagram).
2. The client obtains a signed JWT token from the authorization server.
3. The client initiates a connection to the broker using the JWT token.
4. The Kafka broker verifies the JWT token.
5. If Ranger authorization is enabled for the Kafka service, the claim within the token containing the user ID (principal) is checked against the policies configured for that user in Ranger. The client is only allowed to carry out permitted operations.

Token verification

The broker verifies the tokens by checking the signature of the token as well as the claims found within the token. The broker can verify the following claims:

- “iat” (issued at/timestamp) and “exp” (expiry): Used to verify that the token is valid and has not expired.
- "iss" (issuer): Used to verify that the token was issued by a trusted issuer. The broker only verifies this claim if it is configured to do so. This is done by providing the broker with an “iss” claim value that it should accept. If the broker is not provided with an “iss” claim value, it does not verify the claim.
- “aud” (audience/intended receiver): Used to verify that the token was created specifically for accessing the broker. If the "aud" claim is present in the JWTs presented to the broker, the broker must be provided with the "aud" claim value found within the tokens. Otherwise, the token is considered invalid.

JWKS access

In order for the broker to be able to verify signed JWTs, it must have access to a JWKS. The JWKS is a JSON object that contains the keys which the authorization server uses to sign JWTs. Kafka brokers can be provided access to a JWKS in two different ways. These are as follows:

Note: The two JWKS access options are mutually exclusive. If you configure the broker for both online and local JWKS access, the broker uses local JWKS access.

Online JWKS access

Kafka brokers can retrieve the JWKS directly from the authorization server. If an endpoint is made available to the broker, using appropriate configuration property, the broker connects to that endpoint at startup and fetches the JWKS.

For online JWKS access to work, the broker must be able to access the endpoint through the network. As a result, you might need to set up firewall rules. Additionally, because the broker attempts to retrieve the JWKS at startup, if for example the authorization server is inaccessible or returns an error, the broker will fail to start.

Local JWKS access

If you do not want to rely on the availability of the authorization server to retrieve the JWKS or if establishing a connection between the broker and the authorization server is not possible, you can choose to have the Kafka broker use a JWKS that is available locally on the cluster. For this method to work, you must provide the JWKS JSON directly to the Kafka broker using an advanced configuration snippet.

Configuring OAuth2 authentication for Kafka brokers

Learn how to enable and configure OAuth2 authentication for Kafka brokers.

About this task

You can enable Kafka to use OAuth2 for client to broker authentication. Broker configuration is done by configuring OAuth2 related properties. OAuth2 related properties can be configured using Cloudera Manager. However, two distinct path of configuration exist. Which configuration path you take depends on whether the Kafka service is Kerberos enabled.

If Kerberos is enabled, you can use the OAuth2 related properties directly available in Cloudera Manager to enable and configure OAuth2. If Kerberos is not enabled, you must use an advanced configuration snippet to enable and configure OAuth2.

In addition, when configuring OAuth2 you can choose how the broker gets access to the JSON Web Key Set (JWKS) used to verify tokens. Kafka brokers can be provided access to a JWKS in two different ways. They can be configured to retrieve the JWKS from the authorization server JWKS endpoint (HTTPS) at startup. This is called online JWKS access. Alternatively, providing the broker directly with the JWKS used by the authorization server is also possible using an advanced configuration snippet. This is called local JWKS access.

Note: The two JWKS access options are mutually exclusive. If you configure the broker for both online and local JWKS access, the broker uses local JWKS access.

Before you begin

- If you want to configure online JWKS access, the broker must be able to open a network connection to the authorization server. Otherwise the broker will fail to start.
- If you want to configure local JWKS access, the JWKS JSON must be acquired from the authorization server.

Procedure

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

3. Enable and configure Oauth2 authentication.

   OAuth2 related properties are configured in either of two ways. If Kerberos is enabled for the Kafka service, you can use the properties directly available in Cloudera Manager. In a case like, this follow the instructions in the Using dedicated properties tab. If Kerberos is not enabled for the Kafka service, you must use an advanced configuration snippet. In a case like this, follow the instruction in the Using an advanced configuration snippet tab.

   For Using dedicated properties

   a. Find and configure the following properties:

   Note: The following properties do not take effect if Kerberos is not enabled for the Kafka service. Follow the instructions in the Using an advanced configuration snippet tab if you do not have Kerberos enabled

   Table 4: OAuth properties

Cloudera Manager Property	Description
Enable OAuth Authentication oauth.enabled	Enables authentication using the SASL OAUTHBEARER mechanism for this Kafka service. Clients must present a signed JSON Web Token (JWT) when authenticating with OAuth2. Kerberos must be enabled for this property to take effect.
JWKS URL For OAuth2 oauth.jwks.url	The endpoint URL that returns the authorization server's public keys in JWKS format. Configure a valid JWKS endpoint URL if you want to configure the broker for online JWKS access. If you want to configure the broker for local JWKS access, leave this property empty and instead configure the Kafka Broker Advanced Configuration Snippet (Safety Valve) for oauth_jwks.json property.
Kafka Broker Advanced Configuration Snippet (Safety Valve) for oauth_jwks.json oauth_jwks.json_role_safety_valve	Specifies the JWKS used by this Kafka service to verify JWTs presented by clients. Add the JWKS JSON you obtained from the authorization server’s JWKS endpoint. Set this property if you want to configure the broker for local JWKS access. If you want to configure the broker for online JWKS access, leave this property empty and instead configure the JWKS URL For OAuth2 propert.
JWT Expected Audience For OAuth2 oauth.expected.audience	The JWT token can optionally contain an"aud" (audience or intended receiver) claim. When this claim is present, the same audience value must be specified for the broker, otherwise the token will be considered invalid. This property specifies what the value of the "aud" claim is. Configuring this property is only required if the “aud” claim is present in the tokens presented to the broker.
JWT Expected Issuer For OAuth2 oauth.expected.issuer	The JWT token can optionally contain an "iss" (issuer) claim. This property specifies what the value of the "iss" claim is. Specifying an "iss" claim configures the broker to only accept tokens issued by a specific issuer. Leave this property empty if you do not want the broker to validate the "iss" claim.
JWT Principal Claim Name For OAuth2 oauth.principal.claim.name	A JWT token must contain a user ID (principal) that can be used for Ranger authorization. This property specifies the name (or key) of the claim in the JWT token that contains the user ID. The "sub" (subject) claim is used if this property is left empty. Configuring this property is mandatory if the JWT token does not include a “sub” claim.

b. Optional: Configure additional properties.

If required, additional OAuth2 related properties can be configured with the Kafka Broker Advanced Configuration Snippet (Safety Valve) for kafka.properties property. See *Broker Configs* in the Apache Kafka documentation for a full list of these properties.

For Using an advanced configuration snippet

a. Find and configure the Kafka Broker Advanced Configuration Snippet (Safety Valve) for kafka.properties property.

Use the following example as a template and make changes as necessary. The following example contains all minimum required OAuth2 properties. If required, additional OAuth2 related properties can be configured as well. See *Broker Configs* in the Apache Kafka documentation for a full list of these properties.

Example configuration:

```
listeners=PLAINTEXT://:9092,SASL_PLAINTEXT://:34343
listener.name.sasl_plaintext.sasl.enabled.mechanisms=OAUTHBEARER
listener.name.sasl_plaintext.oauthbearer.sasl.server.callback.handl
er.class=org.apache.kafka.common.security.oauthbearer.secured.OAuthB
earerValidatorCallbackHandler
listener.name.sasl_plaintext.oauthbearer.sasl.jaas.config=org.apac
he.kafka.common.security.oauthbearer.OAuthBearerLoginModule required;
sasl.oauthbearer.jwks.endpoint.url=https://my-authorization-server/
jwks
```

4. Configure the allow list of OAuth URLs.

   You can restrict which OAuth URLs Kafka is allowed to connect to by configuring the `org.apache.kafka.sasl.oauthbearer.allowed.urls` Java option. By specifying a trusted list of OAuth servers, you can have stricter control over the OAuth servers Kafka can access.

   a) Find the Additional Broker Java Options property.
   b) Add the `org.apache.kafka.sasl.oauthbearer.allowed.urls` Java option. For example:

   ```
   -Dorg.apache.kafka.sasl.oauthbearer.allowed.urls=HTTP://WWW.OAUTH-
   EXAMPLE-1.COM,HTTP://WWW.OAUTH-EXAMPLE-2.COM
   ```

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

Results

The broker is ready to accept connections authenticated with OAuth2. Depending on how you configured the broker, it either retrieves the JWKS from the authorization server at startup, or loads the JWKS keys from local storage.

What to do next

Configure your Kafka clients to use OAuth2 authentication. Continue with *Configuring OAuth authentication for Kafka clients*.

Related Information

Broker Configs

Configuring OAuth2 authentication for Kafka clients

OAuth2 configuration examples

Configuring OAuth2 authentication for Kafka clients

Learn how to enable and configure OAuth authentication for Kafka clients.

About this task

In order for the clients to successfully establish a connection with a broker that has OAuth2 authentication enabled, you must configure OAuth2 related properties for the client. Configuring these properties instructs the client to acquire a signed JSON Web Token (JWT) and present that token to the broker when requesting access.

Note: Cloudera Kafka only supports the client_credentials grant type for access tokens.

The following steps demonstrate configuration for console clients. This is done by creating a client.properties configuration file that includes the required properties. If you are configuring a custom developed client, see *Java client security examples* or *.Net client security examples* for code examples.

Before you begin

- Ensure that the authorization server is reachable from the client host.
- Obtain the authorization server's token endpoint URL.
- Obtain the necessary credentials (client ID, client secret) and other parameters (for example, scope) from the authorization server.
- Cloudera recommends that you configure your authorization servers as well as your Kafka brokers to use TLS/SSL. This is recommended so that JWT tokens and client credentials do not get exposed.

Procedure

1. Create a client.properties file containing the following properties:

```
security.protocol=[***SECURITY PROTOCOL***]
sasl.mechanism=OAUTHBEARER
sasl.login.callback.handler.class=org.apache.kafka.common.security.oauth
bearer.secured.OAuthBearerLoginCallbackHandler
sasl.oauthbearer.token.endpoint.url=http://[***OAUTH SERVER***]/[***TOKEN
 ENDPOINT***]
```

   Replace *[***SECURITY PROTOCOL***]* with either SASL_SSL or SASL_PLAINTEXT. The security protocol you specify depends on whether TLS/SSL encryption is enabled on the broker.

2. If TLS/SSL is enabled on the broker, add all required TLS/SSL properties to the client.properties file. For example:

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

   This example contains the minimum required TLS/SSL properties. Depending on your requirements and how TLS/SSL is configured on the broker, other properties might be required. For more information regarding TLS/SSL configuration, see *Channel Encryption*.

3. Configure the JAAS.

   You have two options when configuring the JAAS. You can either embed the full JAAS configuration in the client.properties file or use a separate JAAS configuration file.

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

   ```
   sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthB
   earerLoginModule required clientId="[***CLIENT ID***]" clientSecr
   et="[***CLIENT SECRET***]" scope="[***SCOPE***]";
   ```

   b) Use a separate JAAS config file:

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

      You can also create a new JAAS configuration file if you do not have an existing one available.

      ```
      KafkaClient {
        org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginMod
      ule required
        clientId="[***CLIENT ID***]"
        clientSecret="[***CLIENT SECRET***]"
        scope="[***SCOPE***]";
      };
      ```

   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***]"
      ```

Results

Clients that you run with the configuration file you created authenticate themselves to the Kafka broker with OAuth2.

What to do next

Run a client:

Console consumer

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

Console producer

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

Related Information

Java client security examples

.Net client security examples

Channel encryption

OAuth2 configuration examples

Kafka client configuration reference

This topic collects the minimum configuration required for enable the supported authentication mechanisms. You may need to update the client configuration file and the JAAS, depending on the desired authentication mechanism.

Note: This section does contain the full details for configuring the client for each authentication mechanism. For more details, see *Configure Kafka clients* for the desired authentication mechanism, *Enable Kerberos authentication for Kafka* for Kerberos authentication, or *Client authentication using delegation tokens* for delegation token authentication.

Client configuration file

You need to configure at least the following properties in the client.configuration file to enable the authentication mechanisms supported by Kafka.

For OAuth

```
security.protocol=[***SECURITY PROTOCOL***]
sasl.mechanism=OAUTHBEARER
sasl.login.callback.handler.class=org.apache.kafka.common.securi
ty.oauthbearer.secured.OAuthBearerLoginCallbackHandler
sasl.oauthbearer.token.endpoint.url=http://[***OAUTH
 SERVER***]/[***TOKEN ENDPOINT***]
```

Replace *[***SECURITY PROTOCOL***]* with either SASL_SSL or SASL_PLAINTEXT. The security protocol you specify depends on whether TLS/SSL encryption is enabled on the broker.

The following properties are only required if TLS/SSL is enabled on the broker.

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

This example contains the minimum required TLS/SSL properties. Depending on your requirements and how TLS/SSL is configured on the broker, other properties might be required. For more information regarding TLS/SSL configuration, see *Channel Encryption*.

For LDAP

```
sasl.mechanism=PLAIN
```

The following properties are used for configuring 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.

For PAM

```
sasl.mechanism=PLAIN
```

The following properties are used for configuring 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.

For TLS#SSL

```
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***]
```

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

For Kerberos

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

For Delegation token

This authentication method is not configured using the client.configuration file.

JAAS configuration

You need to configure at least the following properties in the JAAS to enable the authentication mechanisms supported by Kafka.

For OAuth

You have two options when configuring the JAAS. You can either embed the full JAAS configuration in the client.properties file or use a separate JAAS configuration file.

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

```
sasl.jaas.config=org.apache.kafka.common.security.oauthbeare
r.OAuthBearerLoginModule required clientId="[***CLIENT ID***]"
 clientSecret="[***CLIENT SECRET***]" scope="[***SCOPE***]";
```

- Use a separate JAAS configuration file:

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

     You can also create a new JAAS configuration file if you do not have an existing one available.

     ```
     KafkaClient {
       org.apache.kafka.common.security.oauthbearer.OAuthBearer
     LoginModule required
       clientId="[***CLIENT ID***]"
       clientSecret="[***CLIENT SECRET***]"
       scope="[***SCOPE***]";
     };
     ```

  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.co
     nfig=[***PATH TO JAAS.CONF***]"
     ```

For LDAP

You have two options when configuring the JAAS. You can either embed the full JAAS configuration in the client.properties file or use a separate JAAS configuration file.

- 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 req
  uired \
      username="[***USERNAME***]" \
      password="[***PASSWORD***]";
  ```

  Replace *[***USERNAME***]* and *[***PASSWORD***]* with the credentials of the Cloudera 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.
- Use a separate JAAS configuration file:

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

     You can also create a new JAAS configuration file if you do not have an existing one available.

     Example configuration:

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

     Replace *[***USERNAME***]* and *[***PASSWORD***]* with the credentials of the Cloudera 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.co
nfig=[***PATH TO JAAS.CONF***]"
```

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

For PAM

You have two options when configuring the JAAS. You can either embed the full JAAS configuration in the client.properties file or use a separate JAAS configuration file.

- 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 req
uired \
    username="[USERNAME]" \
    password="[PASSWORD]";
```

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

- Use a separate JAAS configuration file:

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

   You can also create a new JAAS configuration file if you do not have an existing one available.

   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_T
O_JAAS.CONF]"
```

   Replace *[PATH_TO_JAAS.CONF]* with the location of the JAAS configuration file you created.

For TLS#SSL

This step is not applicable for TLS/SSL authentication.

For Kerberos

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";
};
```

For Delegation token

You have two options when configuring the JAAS. You can either embed the full JAAS configuration in the client.properties file or use a separate JAAS configuration file.

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

  Example Configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.scram.Scra
mLoginModule  required \
    username="TOKENID" \
    password="LAYYSFMLS4BTJF+LTZ1LCHR/ZZFNA==" \
    tokenauth="TRUE";
security.protocol=SASL_SSL
sasl.mechanism=SCRAM-SHA-256
```

  Within the JAAS configuration, 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.
- Use a separate JAAS configuration file:

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

     You can also create a new JAAS configuration file if you do not have an existing one available.

     The module has to specify the username, password and tokenauth options. The configuration is similar to the following example:

```
KafkaClient {
org.apache.kafka.common.security.scram.ScramLoginModule  r
equired
    username="TOKENID"
    password="LAYYSFMLS4BTJF+LTZ1LCHR/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. Configure the following properties for your clients.

     These properties are added to producer.properties or consumer.properties file that the client uses.

```
security.protocol=SASL_SSL
sasl.mechanism=SCRAM-SHA-256
```

3. 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.co
   nfig=[PATH_TO_JAAS.CONF]"
   ```

Related Information

Configure TLS/SSL authentication for Kafka clients
Client authentication using delegation tokens
Configure Kafka LDAP authentication for Kafka clients
Configure PAM authentication for Kafka clients
Configuring OAuth2 authentication for Kafka clients
Enable Kerberos authentication for Kafka

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 Cloudera*.

Related Information

Using Ranger to Provide authorization in Cloudera

Enable authorization in Kafka with Ranger

Learn how to enable Ranger authorization for Kafka.

About this task

The following instructions can be used to enable and configure Ranger authorization for Kafka where the Ranger service is either deployed on the same cluster as the Kafka service or if the Ranger service is deployed in a Data Context cluster.

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
- It is also possible to have a Kafka service depend on a Ranger service which is deployed on a remote, non Data Context cluster. This is achieved by configuring the following advanced configuration snippets instead of the configuration steps described below:
  - Kafka Broker Advanced Configuration Snippet (Safety Valve) for ranger-kafka-security.xml

    ```
    Name: ranger.plugin.kafka.policy.rest.url
    ```

```
Value: http://[****FQDN OF RANGER ADMIN HOST***]:6080/
```

- Kafka Broker Advanced Configuration Snippet (Safety Valve) for ranger-kafka-audit.xml

```
Name: xasecure.audit.destination.solr.zookeepers
Value: [***FQDN OF ZOOKEEPER HOST***]:2181/solr-infra
```

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 Cloudera 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

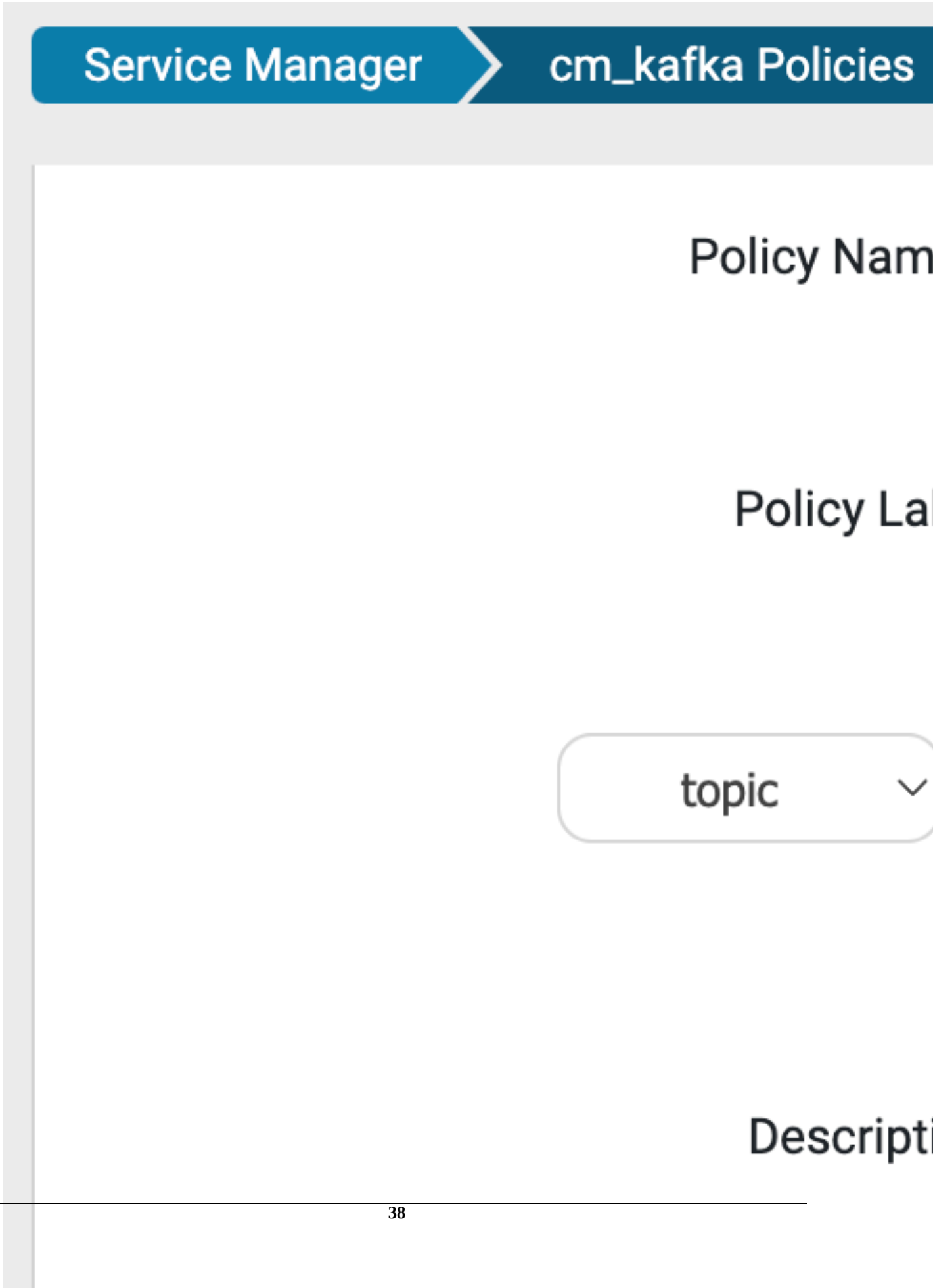
KRaft Ranger authorization

Learn how KRaft integrates with Ranger as well as the default policies and permissions set up for KRaft.

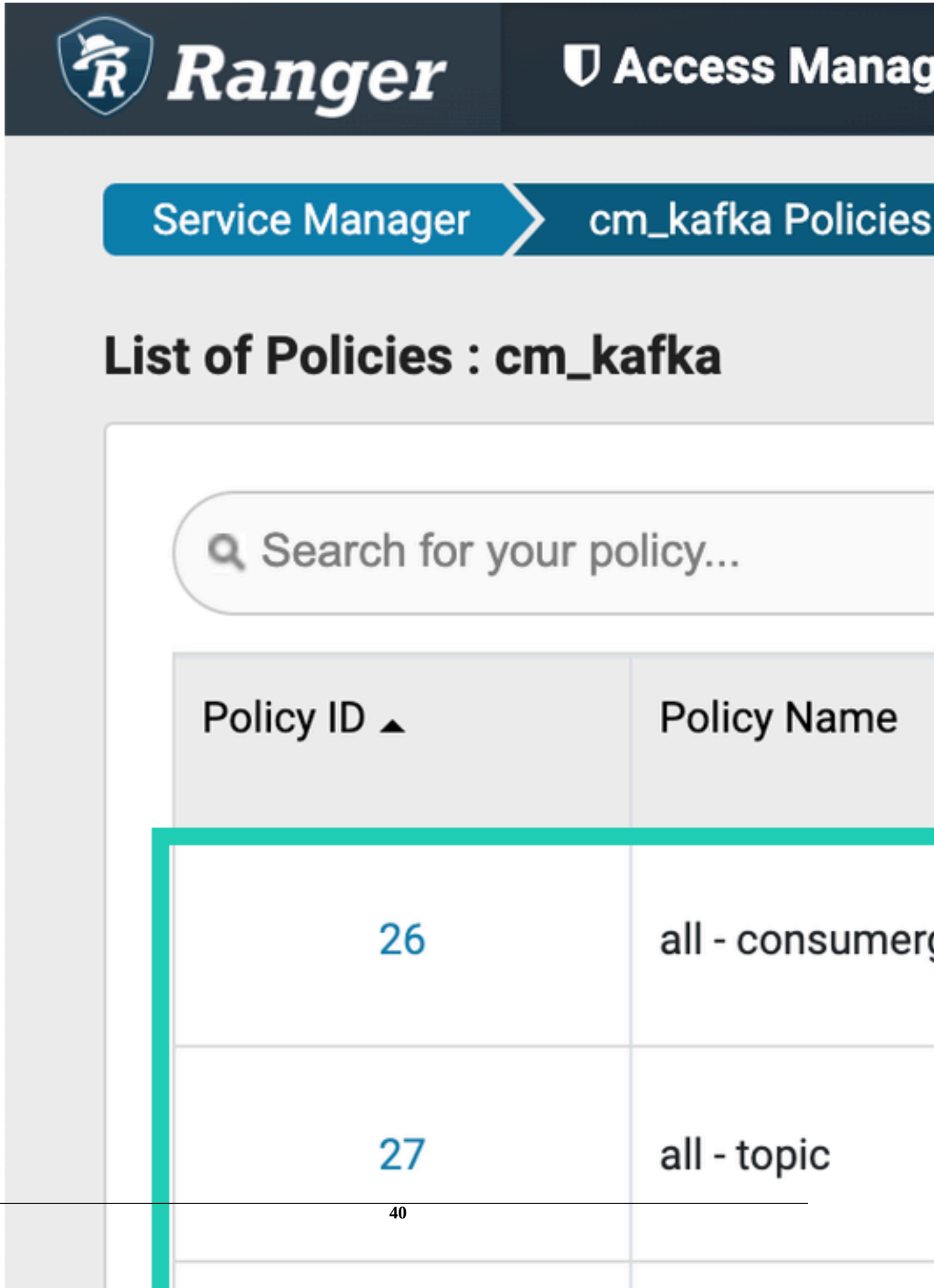
Note: If Ranger authorization is enabled, Kafka still connects to ZooKeeper for auditing. As a result, Kafka's JAAS configuration includes a client entry for ZooKeeper. Additionally, the -Dzookeeper.sasl.client.user name=*[**ZOOKEEPER PRINCIPAL SHORTNAME***]* system property is set for the process. This is the result of Ranger's dependency on ZooKeeper. Even though Ranger makes this connection, Kafka does not require or use ZooKeeper for metadata management if it is running in KRaft mode.

KRaft in Cloudera uses the `KafkaRangerAuthorizer` to authorize requests coming from other entities. In KRaft mode, Kafka brokers forward requests to the controllers and the controllers authorize these requests.

Kraft Controllers run as the kraft user. By default, the Kafka resource-based service in Ranger includes a kraft in ternal - topic policy. This policy grants all permission on the __cluster_metadata topic for the kraft user as well as Describe, Describe Configs, and Consume permissions for the kafka user (default user for brokers). By default, other users do not have access to the __cluster_metadata topic.


Service Manager
cm_kafka Policies
Policy Nam
Policy La
topic
Descripti

In addition, the kraft user is added to all default Kafka policies that grant all permissions on Kafka resources.


Ranger
Access Manag
Service Manager
cm_kafka Policies
List of Policies : cm_kafka
Search for your policy...
Policy ID
Policy Name
26
all - consumerg
27
all - topic

Kafka ACL APIs support in Ranger

Kafka ACL operations in Ranger

Background

Kafka supports various ACL operations, such as APIs like `createAcls`, `deleteAcls`, `acls`, and `authorizeByResourceType`. However, when Ranger is enabled, Kafka does not support:

- grant and revoke of policies on Kafka resources
- list the resources ACLs based on Ranger policies
- check the ACLs on resources

RangerKafkaAuthorizer includes ACL APIs that refer to Ranger Policies when these commands are executed. Ranger relies on the grant, revoke and policy engine APIs to cater the needed functionality.

Note: The following functionality is NOT supported in 7.1.9:

--allow-host and --deny-host

Host name in the API call. This is not supported as this requires grant/revoke ranger API to support the creation of policy conditions for the policy that is getting created.

--deny-principal

Grant except and Revoke except are not supported.

–-operations

Multiple operations on Single command is not supported.

Permission ALL is not supported in the command line.

Reference Commands

List acl

```
kafka-acls --bootstrap-server rm-718ssl-1.rm-718ssl.root.hwx.sit
e:9093 --command-config /kafkatest/kafka-client.conf --list --to
pic connect-configs
```

```
kafka-acls --bootstrap-server rm-718ssl-1.rm-718ssl.root.hwx.sit
e:9093 --command-config /kafkatest/kafka-client.conf --list --cl
uster test_cluster1
```

```
afka-acls --bootstrap-server rm-718ssl-1.rm-718ssl.root.hwx.site
:9093 --command-config /kafkatest/kafka-client.conf --list --clu
ster
```

Create acl

```
kafka-acls --bootstrap-server rm-718ssl-1.rm-718ssl.root.hwx.sit
e:9093 --command-config /kafkatest/kafka-client.conf --add --all
ow-principal User:testuser1 --operation read --topic finance-top
ic
```

```
kafka-acls --bootstrap-server rm-718ssl-1.rm-718ssl.root.hwx.sit
e:9093 --command-config /kafkatest/kafka-client.conf --add --all
ow-principal Group:mysql --operation read --topic finance-topic
```

```
kafka-acls --bootstrap-server rm-718ssl-1.rm-718ssl.root.hwx.sit
e:9093 --command-config /kafkatest/kafka-client.conf --add --all
```

```
ow-principal Group:mysql --allow-principal User:testuser1 --oper
ation read --topic finance-topic
```

```
kafka-acls --bootstrap-server rm-718ssl-1.rm-718ssl.root.hwx.sit
e:9093 --command-config /kafkatest/kafka-client.conf --add --all
ow-principal User:testuser1 -operation read --topic finance-topic
 --resource-pattern-type  prefixed
```

Revoke acl

```
kafka-acls --bootstrap-server rm-718ssl-1.rm-718ssl.root.hwx.sit
e:9093 --command-config  /kafkatest/kafka-client.conf --remove --
allow-principal Group:mysql -operation read --topic finance-topic
```

```
kafka-acls --bootstrap-server rm-718ssl-1.rm-718ssl.root.hwx.sit
e:9093 --command-config  /kafkatest/kafka-client.conf --remove -
-allow-principal User:testuser1 -operation read --topic finance-
topic
```

Related Information

Authorization using ACLs: Confluent Developer documentation

Apache Kafka documentation

Governance

You can monitor and govern Kafka metadata in Atlas. This can be done by importing Kafka topics and clients as entities (metadata) into Atlas.

Kafka entities can be imported into Atlas by running a one-time import using the Import Kafka Topics Into Atlas action available in Cloudera Manager or by configuring the Atlas hook in Kafka. The Import Kafka Topics Into Atlas action imports existing entities only. Setting up the Atlas hook enables continuous import of newly created Kafka entities, but does not import existing ones.

Importing Kafka entities into Atlas

Existing Kafka topics and clients can be imported into Atlas as entities using the Import Kafka Topics Into Atlas action available in Cloudera Manager. Importing Kafka entities allows you to manage, govern, and monitor your topics and clients in Atlas.

About this task

The following list of steps walk you through how you can import Kafka topics and clients as entities (metadata) into Atlas using the Import Kafka Topics Into Atlas action in Cloudera Manager. This action only imports existing topics and clients. Topics and clients created after the import is completed are not automatically imported into Atlas.

Continuous import can be enabled by configuring the Atlas hook in Kafka. For more information, see *Configuring the Atlas hook in Kafka*.

Before you begin

- An Atlas service is deployed on the Kafka cluster or the data context cluster. The Atlas service must be started and must be in a healthy state.
- You are logged in to Cloudera Manager as a user that can perform service and role level configuration changes (Cluster Administrator, Full Administrator, Configurator, Limited Cluster Administrator).

- If authorization is enabled, ensure that the kafka user is authorized to create, update, delete, and read Kafka entities in Atlas.

Procedure

1. In Cloudera Manager select the Kafka service.
2. Click ActionsImport Kafka Topics Into Atlas.
3. Click Import Kafka Topics Into Atlas and wait until the process is finished.

Results

Existing Kafka topics and clients imported into the Atlas service.

What to do next

Access the Atlas dashboard to view, monitor, and manage imported Kafka entities.

Related Information

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.
- Import existing Kafka topics into Atlas using the Import Kafka Topics Into Atlas action in Cloudera Manager. For more information, see Importing Kafka entities into Atlas.

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 to 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

Setting up Atlas Kafka import tool

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.

KRaft security

Learn about KRaft security and security configuration in Cloudera.

When you deploy Kafka in KRaft mode a set of specialized broker roles, KRaft Controller roles, are deployed on your cluster. KRaft Controllers communicate with brokers to serve their requests and to manage Kafka's metadata. The connection between controllers and brokers can be secured using TLS/SSL encryption, TLS/SSL authentication, and/or Kerberos authentication.

By default KRaft Controllers inherit the security configuration of the parent Kafka service. For example, if TLS/SSL is enabled for Kafka, then Cloudera Manager automatically enables TLS/SSL for the KRaft Controllers in the cluster. As a result, if you configure security for the Kafka service, no additional configuration is required to secure KRaft Controllers.

However, if required, some security properties related to encryption and authentication can be configured separately for KRaft Controllers.

- TLS/SSL encryption and authentication

  TLS/SSL configuration can be configured separately as the KRaft Controller role has its own set of TLS/SSL properties. You can enable or disable TLS/SSL as well as configure the key and truststore that the KRaft Controller roles use. For more information see, Configure TLS/SSL for KRaft Controllers on page 8.
- Kerberos authentication

  Kerberos cannot be enabled or disabled separately for KRaft Controllers. The default Kerberos principal for KRaft controllers, the kraft user, can be changed using the Role-Specific Kerberos Principal Kafka service property.

  Important: Cloudera Manager configures Cloudera services to use the default Kerberos principal names. Cloudera recommends that you do not change the default Kerberos principal names. If it is unavoidable to do so, contact Cloudera Professional Services because it requires extensive additional custom configuration.

Ranger authorization

In addition to encryption and authentication, the default principal that KRaft Controllers run as is integrated with Ranger. For more information on the default policies set up for the user, see KRaft Ranger authorization on page 37.

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.

Kafka security hardening with Zookeeper ACLs

Access to Kafka metadata in Zookeeper is restricted by default. However, when completing configuration and management tasks for your Kafka service, you might be required to first unlock and then restrict access to metadata in Zookeeper. Continue reading to learn how you can restrict or unlock access to Kafka metadata in Zookeeper.

Restricting access to Kafka metadata in Zookeeper

Learn how to restrict access to Kafka metadata in Zookeeper.

About this task

Locking down Kafka znodes (metadata) 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 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.

   a) In Cloudera Manager select the Kafka service.
   b) Select Configuration.
   c) Find the Enable Zookeeper ACL property.
   d) Set the property to true by selecting 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) Check the Restart roles with stale configurations only checkbox and click Rolling restart.
   c) 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]/[ZOOK
   EEPER_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.

   a) Change the JVMFLAGS environment variable to -Djava.security.auth.login.confi g=*[PATH_TO_JAAS.CONF]*.

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

   b) Start the zookeeper client.

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

   c) Enter the following to reset the ACLs of the root node.

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

6. Verify znode permissions.

   You can do this with the getAcl ZooKeeper CLI command. For example.

   ```
   getAcl [ZK NODE PATH]
   ```

   If everything is set up correctly, running this command on the /kafka znode (default chroot for Kafka) should return the following permissions.

   ```
   'sasl,'kafka
   : cdrwa
   'world,'anyone
   : r
   ```

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

Unlocking access to Kafka metadata in Zookeeper

Learn how to unlock access to Kafka metadata in Zookeeper

About this task

Cloudera does not recommend that you leave access to Kafka znodes (metadata) in Zookeeper unlocked. Only unlock access if you are planning to make configuration changes in Kafka that require znode access to be unlocked. Once configuration changes are complete, Cloudera recommends that you re-restrict access.

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]/[ZOOK
   EEPER_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