Cloudera Runtime 7.0.3

Securing Apache Kafka

Date published: 2019-12-18
Date modified: 2019-12-18

https://docs.cloudera.com/

Legal Notice

Contents

TLS

Learn how to turn on and configure TLS for Kafka.

Kafka allows clients to connect over TLS. By default, TLS is disabled, but can be turned on as needed.

Step 1: Generate keys and certificates for Kafka brokers

Learn how to generate keys and certificates for brokers.

Generate the key and the certificate for each machine in the cluster using the Java keytool utility. For more information see, the Generate TLS Certificates section in Manually Configuring TLS Encryption for Cloudera Manager.

Make sure that the common name (CN) matches the fully qualified domain name (FQDN) of your server. The client compares the CN with the DNS domain name to ensure that it is connecting to the correct server.

Related Information

Manually Configuring TLS Encryption for Cloudera Manager

Step 2: Create your own certificate authority

Learn how to create your own certificate authority.

You have generated a public-private key pair for each machine and a certificate to identify the machine. However, the certificate is unsigned, so an attacker can create a certificate and pretend to be any machine. Sign certificates for each machine in the cluster to prevent unauthorized access.

A Certificate Authority (CA) is responsible for signing certificates. A CA is similar to a government that issues passports. A government stamps (signs) each passport so that the passport becomes difficult to forge. Similarly, the CA signs the certificates, and the cryptography guarantees that a signed certificate is computationally difficult to forge. If the CA is a genuine and trusted authority, the clients have high assurance that they are connecting to the authentic machines.

```
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
```

The generated CA is a public-private key pair and certificate used to sign other certificates.

Add the generated CA to the client truststores so that clients can trust this CA:

```
keytool -keystore {client.truststore.jks} -alias CARoot -import -file {ca-ce
rt}
```

Note: If you configure Kafka brokers to require client authentication by setting ssl.client.auth to be requested or required on the Kafka brokers config, you must provide a truststore for the Kafka brokers as well. The truststore must have all the CA certificates by which the clients keys are signed. The keystore created in step 1 stores each machine's own identity. In contrast, the truststore of a client stores all the certificates that the client should trust. Importing a certificate into a truststore means trusting all certificates that are signed by that certificate. This attribute is called the chain of trust. It is particularly useful when deploying SSL on a large Kafka cluster. You can sign all certificates in the cluster with a single CA, and have all machines share the same truststore that trusts the CA. That way, all machines can authenticate all other machines.

Step 3: Sign the certificate

Learn how to self-sign certificates created for Kafka.

1. Create a certificate request from the keystore:

```
keytool -keystore server.keystore.jks -alias localhost -certreq -fil
e cert-file
```

   where:

   - *keystore*: the location of the keystore
   - *cert-file*: the exported, unsigned certificate of the server
2. Sign the resulting certificate with the CA (in the real world, this can be done using a real CA):

```
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed
 -days validity -CAcreateserial -passin pass:ca-password
```

   where:

   - *ca-cert*: the certificate of the CA
   - *ca-key*: the private key of the CA
   - *cert-signed*: the signed certificate of the server
   - *ca-password*: the passphrase of the CA
3. Import both the certificate of the CA and the signed certificate into the keystore:

```
keytool -keystore server.keystore.jks -alias CARoot -import -file ca-cert
keytool -keystore server.keystore.jks -alias localhost -import -file cert-
signed
```

The following Bash script demonstrates the steps described above. One of the commands assumes a password of SamplePassword123, so either use that password or edit the command before running it.

```
#!/bin/bash
#Step 1
keytool -keystore server.keystore.jks -alias localhost -validity 365 -genkey
#Step 2
openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
keytool -keystore server.truststore.jks -alias CARoot -import -file ca-cert
keytool -keystore client.truststore.jks -alias CARoot -import -file ca-cert
#Step 3
keytool -keystore server.keystore.jks -alias localhost -certreq -file cert-
file
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -
days 365 -CAcreateserial -passin pass:SamplePassword123
keytool -keystore server.keystore.jks -alias CARoot -import -file ca-cert
keytool -keystore server.keystore.jks -alias localhost -import -file cert-s
igned
```

Step 4: Configure Kafka brokers

Learn how to configure TLS/SSL communication for Kafka brokers.

Kafka Brokers support listening for connections on multiple ports. If SSL is enabled for inter-broker communication, both PLAINTEXT and SSL ports are required.

To configure the listeners from Cloudera Manager, perform the following steps:

1. In Cloudera Manager, go to Kafka Instances.
2. Go to Kafka Broker Configurations.

3. In the Kafka Broker Advanced Configuration Snippet (Safety Valve) for Kafka Properties, enter the following information:

```
listeners=PLAINTEXT://kafka-broker-host-name:9092,SSL://kafka-broker-host-name:9093
advertised.listeners=PLAINTEXT://kafka-broker-host-name:9092,SSL://kafka-broker-host-name:9093
```

   where *kafka-broker-host-name* is the FQDN of the broker that you selected from the Instances page in Cloudera Manager. In the above sample configurations we used PLAINTEXT and SSL protocols for the SSL enabled brokers.

   For information about other supported security protocols, seeUsing Kafka's inter-broker security on page 19.
4. Repeat the previous step for each broker.

   The advertised.listeners configuration is needed to connect the brokers from external clients.
5. Deploy the above client configurations and rolling restart the Kafka service from Cloudera Manager.

Kafka CSD auto-generates listeners for Kafka brokers, depending on your SSL and Kerberos configuration. To enable SSL for Kafka installations, do the following:

1. Turn on SSL for the Kafka service by turning on the ssl_enabled configuration for the Kafka CSD.
2. Set security.inter.broker.protocol as SSL, if Kerberos is disabled; otherwise, set it as SASL_SSL.

The following SSL configurations are required on each broker. Each of these values can be set in Cloudera Manager. Be sure to replace this example with the truststore password.

For instructions, see Changing the Configuration of a Service or Role Instance.

```
ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
ssl.keystore.password=SamplePassword123
ssl.key.password=SamplePassword123
ssl.truststore.location=/var/private/ssl/server.truststore.jks
ssl.truststore.password=SamplePassword123
```

Other configuration settings may also be needed, depending on your requirements:

- ssl.client.auth=none: Other options for client authentication are required, or requested, where clients without certificates can still connect. The use of requested is discouraged, as it provides a false sense of security and misconfigured clients can still connect.
- 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 TLS or SSL network protocol. This list is empty by default.
- ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1: Provide a list of SSL protocols that your brokers accept from clients.
- ssl.keystore.type=JKS
- ssl.truststore.type=JKS

Communication between Kafka brokers defaults to PLAINTEXT. To enable secured communication, modify the broker properties file by adding security.inter.broker.protocol=SSL.

For a list of the supported communication protocols, see Using Kafka's inter-broker security on page 19.

Note: Due to import regulations in some countries, Oracle implementation of JCA limits the strength of cryptographic algorithms. If you need stronger algorithms, you must obtain the JCE Unlimited Strength Jurisdiction Policy Files and install them in the JDK/JRE as described in JCA Providers Documentation.

After SSL is configured your broker, logs should show an endpoint for SSL communication:

```
with addresses: PLAINTEXT -> EndPoint(192.168.1.1,9092,PLAINTEXT),SSL -> EndPoint(192.168.1.1,9093,SSL)
```

You can also check the SSL communication to the broker by running the following command:

```
openssl s_client -debug -connect localhost:9093 -tls1
```

This check can indicate that the server keystore and truststore are set up properly.

Note: ssl.enabled.protocols should include TLSv1.

The output of this command should show the server certificate:

```
-----BEGIN CERTIFICATE-----
{variable sized random bytes}
-----END CERTIFICATE-----
subject=/C=US/ST=CA/L=Palo Alto/O=org/OU=org/CN=Franz Kafka
issuer=/C=US/ST=CA/L=Palo Alto
/O=org/OU=org/CN=kafka/emailAddress=kafka@your-domain.com
```

If the certificate does not appear, or if there are any other error messages, your keystore is not set up properly.

Related Information

Using Kafka's inter-broker security

Changing the Configuration of a Service or Role Instance

Step 5: Configure Kafka clients

Learn how to configure TLS/SSL communication for Kafka clients.

SSL is supported only for the new Kafka producer and consumer APIs. The configurations for SSL are the same for both the producer and consumer.

Before you begin with configuration, you need to acquire keys and signed certificates for all clients.

If client authentication is not required in the broker, the following example shows a minimal configuration:

```
security.protocol=SSL
ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks
ssl.truststore.password=SamplePassword123
```

If client authentication is required, a keystore must be created as well and it needs to be signed by a CA. In addition, the following properties: must be configured:

```
ssl.keystore.location=/var/private/ssl/kafka.client.keystore.jks
ssl.keystore.password=SamplePassword123
ssl.key.password=SamplePassword123
```

Other configuration settings might also be needed, depending on your requirements and the broker configuration:

- ssl.provider (Optional). The name of the security provider used for SSL connections. Default is the default security provider of the JVM.
- ssl.cipher.suites (Optional). 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 TLS or SSL network protocol.
- ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1. This property should list at least one of the protocols configured on the broker side.
- ssl.truststore.type=JKS
- ssl.keystore.type=JKS

Authentication

Enable Kerberos authentication

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. See Enabling Kerberos Authentication for Runtime for detailed instructions

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

Procedure

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

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

   - If you use a keytab, use this configuration:

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

     For more information on generating keytabs, see Get or create a Kerberos principal for each user account.
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 your topic exists.

    This does not use security features, but it is a best practice.

    ```
    kafka-topics --list --zookeeper zkhost:2181
    ```

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

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

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 --new-consumer --topic test1 --from-beginning --b
    ootstrap-server anybroker:9092 --consumer.config client.properties
    ```

Related Information

Enabling Kerberos Authentication

Get or create a Kerberos principal for each user account

Delegation token based authentication

An overview on Kafka delegation tokens.

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

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

Many Hadoop components use delegation tokens to mitigate these problems:

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

Kafka Delegation Token Basics

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

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

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

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

   Renewed

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

   Revoked

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

Broker Configuration

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

Related Information

HMAC Wikipedia Page

Client authentication using delegation tokens

Kafka Properties in Cloudera Runtime

Enable or disable authentication with delegation tokens

Learn how to enable or disable Kafka delegation tokens.

About this task

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

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

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

Delegation tokens can be enabled or disabled 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

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 Zookeeper.

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

Procedure

Configure the JAAS configuration property for each client.

Example Configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule  re
quired \
    username="tokenID" \
    password="lAYYSFmLs4bTjf+lTZ1LCHR/ZZFNA==" \
    tokenauth="true";
```

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

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

Configure clients on an application level

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

About this task

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

Procedure

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

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

   Example Configuration:

   ```
   KafkaClient {
   org.apache.kafka.common.security.scram.ScramLoginModule  required
       username="tokenID"
       password="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. Pass the location of your JAAS configuration file as a JVM parameter through a command line interface.

   This sets the JAAS configuration on the Java process level.

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

Kafka security hardening with Zookeeper ACLs

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

Restrict access to Kafka metadata in Zookeeper

Learn how to restrict access to Kafka metadata in Zookeeper.

About this task

Locking down znodes in Zookeeper can be used to protect Kafka metadata against unauthorized access. Direct manipulation of metadata in Zookeeper is not only dangerous for the health of the cluster, but can also serve as an entry point for malicious users to gain elevated access who can then alter the owner or renewer of delegation tokens.

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 zookeeper.set.acl configuration parameter to true.

   a) In Cloudera Manager select the Kafka service.
   b) Select Configuration and find the Kafka Broker Advanced Configuration Snippet (Safety Valve) for kafka.properties property.
   c) Add the following line to the Kafka Broker Advanced Configuration Snippet (Safety Valve) for kafka.properties property:

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

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 config 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"
   ```

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

   ```
   zookeeper-security-migration --zookeeper.connect hostname:port --zookeeper
   .acl secure
   ```

   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.config=*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
   ```

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

Unlock Kafka metadata in Zookeeper

Learn how to unlock access to Kafka metadata in Zookeeper

Before you begin

A secure Kafka cluster with Kerberos authentication enabled is required.

Procedure

1. Disable the use of secure ACLs by setting zookeeper.set.acl configuration parameter to false:
   a) In Cloudera Manager select the Kafka service.
   b) Select Configuration and find the Kafka Broker Advanced Configuration Snippet (Safety Valve) for kafka.properties property.
   c) Add the following line to the Kafka Broker Advanced Configuration Snippet (Safety Valve) for kafka.properties property:

   ```
   zookeeper.set.acl=false
   ```

2. Perform a Rolling Restart:
   a) Return to the Home page by clicking the Cloudera Manager logo.
   b) Go to the Kafka service and select Actions Rolling Restart.
   c) Check the Restart roles with stale configurations only checkbox and click Rolling restart.
   d) Click Close when the restart has finished.
3. Run the zookeeper-security-migration tool with the zookeeper.acl option set to unsecure

   ```
   zookeeper-security-migration --zookeeper.connect hostname:port --zookeeper
   .acl unsecure
   ```

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

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 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.

Procedure

1. In Cloudera Manager select the Kafka service.
2. Select Configuration.
3. Enable LDAP authentication:
   a) Find the Enable LDAP Authentication property.
   b) Enable the property for all required Kafka services by checking the checkbox next to the name of the Kafka service.
4. Configure the LDAP URL:
   a) FInd the LDAP URL property.
   b) Add your LDAP server URL.
   For example:
   ```
   ldap://cloudera.example.com:636
   ```
   Note: Depending on how your LDAP is set up, the URL can start with either ldap:// or ldaps://.
5. Configure the LDAP User DN Template:
   a) Find the LDAP User DN Template property.
   b) Add your template.
   For example:
   ```
   uid={0},cn=users,cn=account,dc=cloudera,dc=example,dc=com
   ```
6. Click Save Changes.
7. 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 clients

Configure 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.

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="[LDAP_USERNAME]" \
       password="[LDAP_PASSWORD]";
   ```

   Replace *[LDAP_USERNAME]* and *[LDAP_PASSWORD]* with a valid LDAP 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="[LDAP_USERNAME]"
           password="[LDAP_PASSWORD]";
          };
      ```

      Replace *[LDAP_USERNAME]* and *[LDAP_PASSWORD]* with a valid LDAP username and password.

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

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

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

Results
LDAP authentication is configured for the client.

Authorization

Ranger

Learn more about authorization with Ranger.

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

Related Information
Using Ranger to Provide Authorization in CDP

Enable authorization in Kafka with Ranger

Learn how to enable Ranger authorization for Kafka.

Before you begin

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

- Kerberos
- LDAP
- SSL

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
TLS
LDAP 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 by default is set to cm_kafka. 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.

Before you begin

Make sure that the resource-based service you want to use exists within Ranger. If not, create it via the Ranger Console's Service Manager page.

Procedure

1. In Cloudera Manager select the Kafka service.
2. Select Configuration and find the Ranger service name for this Kafka cluster property.
3. Enter the name of the resource-based service that you want to use for authorization.
4. Click Save Changes.
5. Restart the Kafka service.

Results

The selected resource-based service is configured and will be used by Kafka for authorization.

Related Information

Apache Ranger Authorization

Using Kafka's inter-broker security

Learn about Kafka inter-broker-security and supported security protocols.

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 broker-to-broker communications. Set the Kafka inter-broker communication protocol using the security.inter.broker.protocol property. Use this property primarily for the following scenarios:

- Enabling SSL encryption for client-broker communication but keeping broker-broker communication as PLAINTEXT. Because SSL has performance overhead, you might want to keep inter-broker communication as PLAINTEXT if your Kafka brokers are behind a firewall and not susceptible to network snooping.

- Migrating from a non-secure Kafka configuration to a secure Kafka configuration without requiring downtime. Use a rolling restart and keep security.inter.broker.protocol set 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, follow these steps:

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

The following combination of protocols are supported.

	SSL	Kerberos
PLAINTEXT	No	No
SSL	Yes	No
SASL_PLAINTEXT	No	Yes
SASL_SSL	Yes	Yes

These protocols can be defined for broker-to-client interaction and for broker-to-broker interaction. The property security.inter.broker.protocol allows the broker-to-broker communication protocol to be different than the broker-to-client protocol, allowing rolling upgrades from non-secure to secure clusters. In most cases, set security.inter.broker.protocol to the protocol you are using for broker-to-client communication. Set security.inter.broker.protocol to a protocol different than the broker-to-client protocol only when you are performing a rolling upgrade from a non-secure to a secure Kafka cluster.