Cloudera Runtime 7.3.2

# Schema Registry Security

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

# CLOUDERA

# Legal Notice

# Contents

# TLS encryption for Schema Registry

Transport Layer Security (TLS) is an industry standard set of cryptographic protocols for securing communications over a network. To encrypt sensitive information between the Cloudera Manager Server and cluster hosts, you must enable TLS.

You can choose to enable Auto-TLS or manually configure TLS.

Auto-TLS simplifies the process of enabling and managing TLS encryption on your cluster. When you enable Auto-TLS, an internal certificate authority (CA) is created and certificates are deployed automatically across all cluster hosts. For more information on Auto-TLS, see *Configuring TLS Encryption for Cloudera Manager Using Auto-TLS*.

If you choose to enable TLS manually, you must create the TLS certificates, making sure the certificates meet the requirements. Then configure Cloudera Manager and Schema Registry.

**Related Information**
Configuring TLS Encryption for Cloudera Manager Using Auto-TLS

## TLS certificate requirements and recommendations

If you choose to manually configure TLS, then you must use your own certificates. The certificates must meet the requirements listed here.

### Certificate requirements

Ensure that the following minimum requirements are satisfied:

- The KeyStore must contain only one PrivateKeyEntry. Using multiple private keys in one KeyStore is not supported.
- The KeyStore password and key/certificate password must be the same or no password should be set on the certificate.
- The unique KeyStores used on each cluster node must use the same KeyStore password and key/certificate password. Ambari and Cloudera Manager do not support defining unique passwords per host.
- The X509v3 ExtendedKeyUsages section of the certificate must have the following attributes:

    - clientAuth

      This attribute is for TLS web client authentication.
    - serverAuth

      This attribute is for TLS web server authentication.
- The signature algorithm used for the certificate must be sha256WithRSAEncryption (SHA-256).
- The certificates must not use wildcards. Each cluster node must have its own certificate.
- Subject Alternate Names (SANs) are mandatory and should at least include the FQDN of the host.
- Additional names for the certificate/host can be added to the certificate as SANs.

    - Add the FQDN used for the CN as a DNS SAN entry.
    - If you are planning to use a load balancer, include the FQDN for the load balancer as a DNS SAN entry.
- The X509v3 KeyUsage section of the certificate must include the following attributes:

    - DigitalSignature
    - Key_Encipherment

### Cloudera recommendations

Cloudera recommends the following security protocols:

- Use certificates that are signed by a CA. Do not issue self-signed certificates.
- Generate a unique certificate per host.

# Configuring TLS encryption manually for Schema Registry

If you do not want to enable Auto-TLS, because, for example, you need to use your own enterprise-generated certificates, you can manually enable TLS for Schema Registry.

### Before you begin

Ensure that you have set up TLS for Cloudera Manager:

1. Review the requirements and recommendations for the certificates.
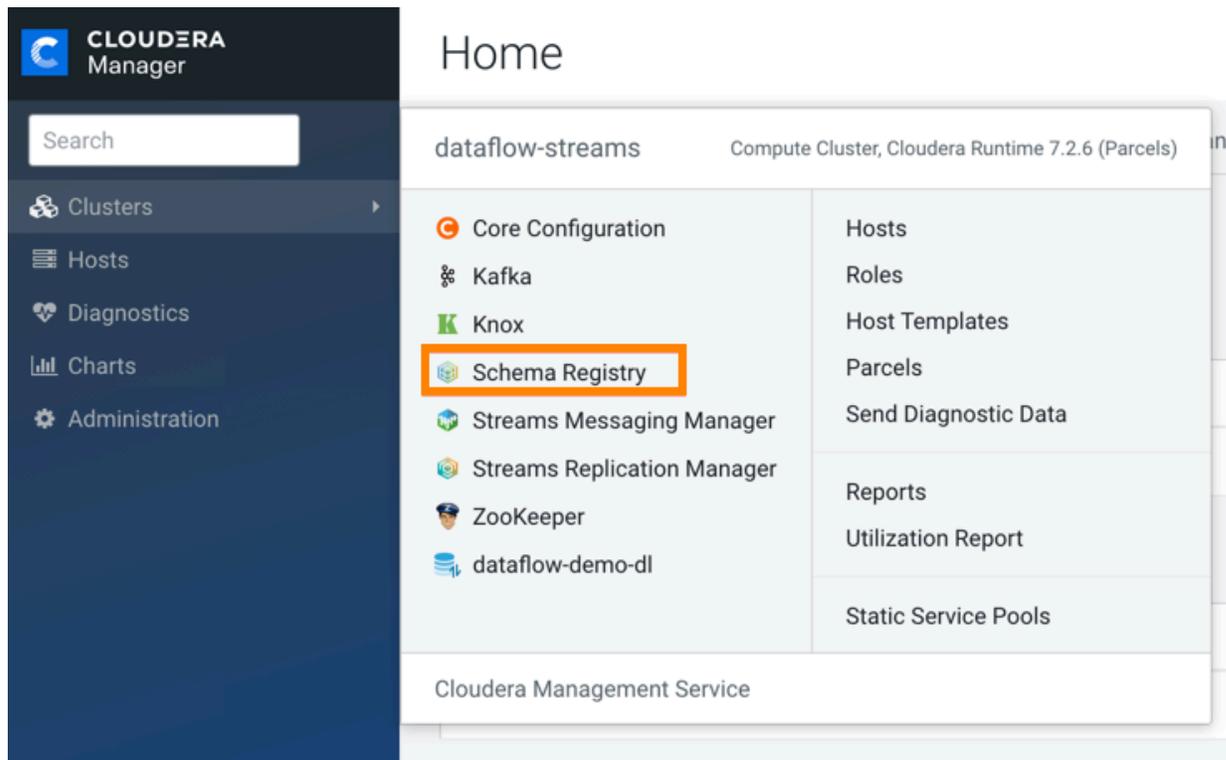
   For more information, see *TLS Certificate Requirements and Recommendations*.
2. Generate the TLS certificates and configure Cloudera Manager.

   For more information, see *Manually Configuring TLS Encryption for Cloudera Manager*.

### Procedure

1. In Cloudera Manager, click  Cluster Schema Registry .



2. Click **Configuration**.
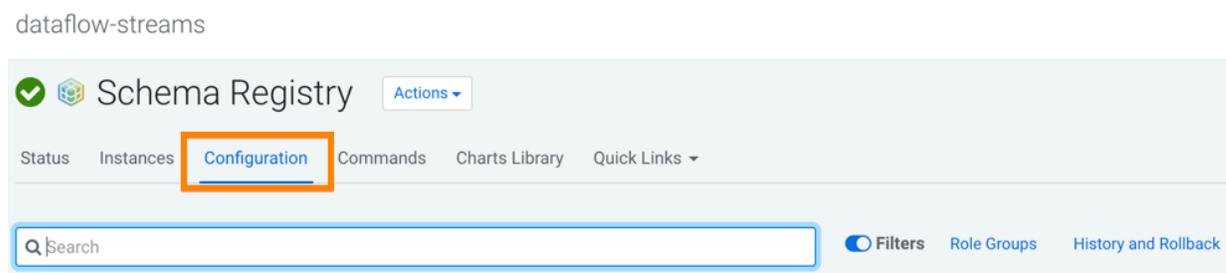
**3.** Enter ssl in the Search field.

The security properties for Schema Registry appear.

**4.** Edit the security properties.

For example:



**5.** Click Save Changes.

**6.** Restart the Schema Registry service.

**Related Information**

TLS certificate requirements and recommendations

Manually Configuring TLS Encryption for Cloudera Manager

# Schema Registry TLS properties

To enable and configure TLS manually for Schema Registry, edit the security properties according to the cluster configuration.

The following table lists the Security properties for Schema Registry:

| Property | Description |
|---|---|
| Schema Registry Port (SSL) <br><br> `schema.registry.ssl.port` | HTTPS port Schema Registry node runs on when SSL is enabled. |
| Schema Registry Admin Port (SSL) <br><br> `schema.registry.ssl.adminPort` | HTTPS admin port Schema Registry node runs on when SSL is enabled. |
| SSL KeyStore Type <br><br> `schema.registry.ssl.keyStoreType` | The keystore type. It is blank by default, but required if Schema Registry's SSL is enabled. For example, PKCS12 or JKS. If it is left empty, then this keystore type is derived from the Cloudera Manager settings. |
| SSL TrustStore Type <br><br> `schema.registry.ssl.trustStoreType` | The truststore type. It is blank by default, but required if Schema Registry's SSL is enabled. For example, PKCS12 or JKS. If it is left empty, then this truststore type is derived from the Cloudera Manager settings. |

| Property | Description |
|---|---|
| SSL ValidateCerts<br><br>`schema.registry.ssl.validateCerts` | Whether or not to validate TLS certificates before starting. If enabled, it does not start with expired or otherwise invalid certificates. |
| SSL ValidatePeers<br><br>`schema.registry.ssl.validatePeers` | Whether or not to validate TLS peer certificates. |
| Version of oracle.net.ssl<br><br>`schema.registry.oracle.net.ssl_version` | Oracle net SSL version. |
| Oracle TLS javax.net.ssl.keyStore<br><br>`schema.registry.javax.net.ssl.keyStore` | Path to keystore file if enabling TLS using Oracle DB. |
| Oracle TLS javax.net.ssl.keyStoreType<br><br>`schema.registry.javax.net.ssl.keyStoreType` | KeyStoreType type if enabling TLS using Oracle DB. |
| Oracle TLS javax.net.ssl.keyStorePassword<br><br>`schema.registry.javax.net.ssl.keyStorePassword` | KeyStorePassword if enabling TLS using Oracle DB. |
| Oracle TLS javax.net.ssl.trustStore<br><br>`schema.registry.javax.net.ssl.trustStore` | Required Path to truststore file if enabling TLS using Oracle DB. |
| Oracle TLS javax.net.ssl.trustStoreType<br><br>`schema.registry.javax.net.ssl.trustStoreType` | Required Truststore type if enabling TLS using Oracle DB. |
| Oracle TLS javax.net.ssl.trustStorePassword<br><br>`schema.registry.javax.net.ssl.trustStorePassword` | TrustStorePassword type if enabling TLS using Oracle DB. |
| Oracle TLS oracle.net.ssl_cipher_suites<br><br>`schema.registry.oracle.net.ssl_cipher_suites` | Oracle net SSL cipher suites if enabling TLS using Oracle DB, for example SSL_DH_DSS_WITH_DES_CBC_SHA. |
| Oracle TLS oracle.net.ssl_server_dn_match<br><br>`schema.registry.oracle.net.ssl_server_dn_match` | Oracle SSL server domain name match if enabling TLS using Oracle DB. |

| Property | Description |
|---|---|
| Enable TLS/SSL for Schema Registry Server<br><br>`ssl.enable` | Encrypt communication between clients and Schema Registry Server using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)). |
| Schema Registry Server TLS/SSL Server JKS Keystore File Location<br><br>`schema.registry.ssl.keyStorePath` | The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Schema Registry Server is acting as a TLS/SSL server. |
| Schema Registry Server TLS/SSL Server JKS Keystore File Password<br><br>`schema.registry.ssl.keyStorePassword` | The password for the Schema Registry Server keystore file. |
| Schema Registry Server TLS/SSL Client Trust Store File<br><br>`schema.registry.ssl.trustStorePath` | The location on disk of the truststore, in .jks format, used to confirm the authenticity of TLS/SSL servers that Schema Registry Server might connect to. This is used when Schema Registry Server is the client in a TLS/SSL connection. This truststore 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. |
| Schema Registry Server TLS/SSL Client Trust Store Password<br><br>`schema.registry.ssl.trustStorePassword` | The password for the Schema Registry Server TLS/SSL certificate TrustStore file. This password is not required to access the truststore; 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. |

## Configuring mutual TLS for Schema Registry

Schema Registry supports two-way TLS authentication, also known as mutual TLS or mTLS. Learn how to configure mutual TLS for Schema Registry.

### About this task

In one-way (or regular) TLS, the server certificate is validated by the client to check if the server can be trusted. Two-way TLS authentication allows both the client and the server to validate each other by both parties sending their respective certificates to the other side.
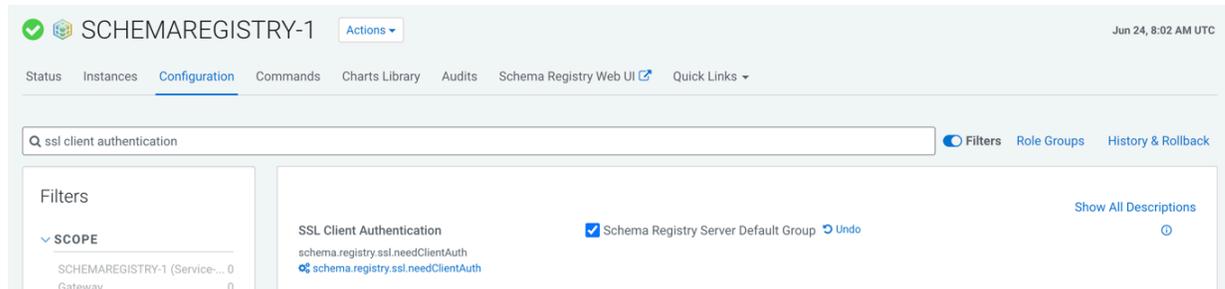
### Before you begin
You must have enabled server-side TLS as described in *Configuring TLS Encryption Manually for Schema Registry*.

### Procedure

1. Go to your cluster in Cloudera Manager.
2. Select Schema Registry from the list of services.
3. Go to the Configuration tab.

**4.** Search for the SSL Client Authentication property, and select the checkbox.



**5.** Click Save Changes.

## What to do next

mTLS integrates with Ranger authorization as well. The TLS certificate sent by the client contains a subject field which has a string value. By default, the contents of this value is passed to Ranger which performs authorization against it.

The value of the subject field may have a complex value and you might want to use a regular expression to extract the value of the principal. In this case you can set the rules in Cloudera Manager.

**1.** Go to your cluster in Cloudera Manager.
**2.** Select Schema Registry from the list of services.
**3.** Go to the Configuration tab.
**4.** Search for the SSL Client Authentication Mapping Rules property, and set it.
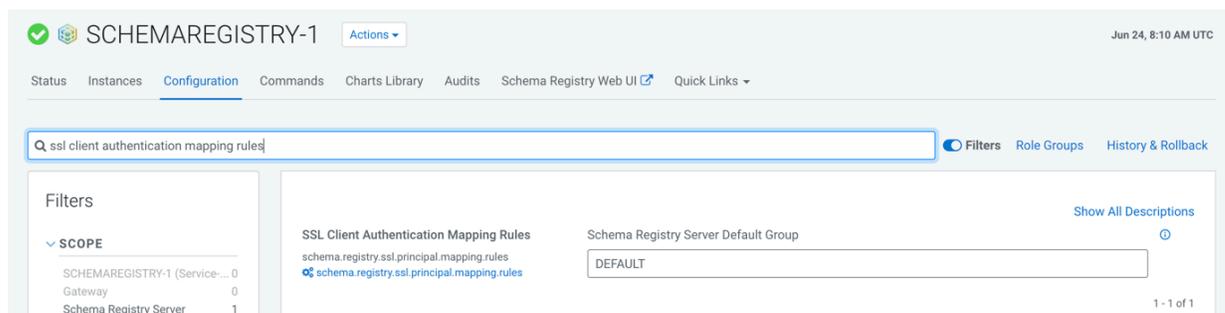


The mapping rules are in the same format as used for Kerberos principals. For more information, see:https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=89071740.

**5.** Search for the Schema Registry Kerberos Name Rules property, and set it.

For example, the Schema Registry Kerberos Name Rules property is set to RULE:^CN=(.*?),OU=ServiceUsers.*$/$1/L in the following image:
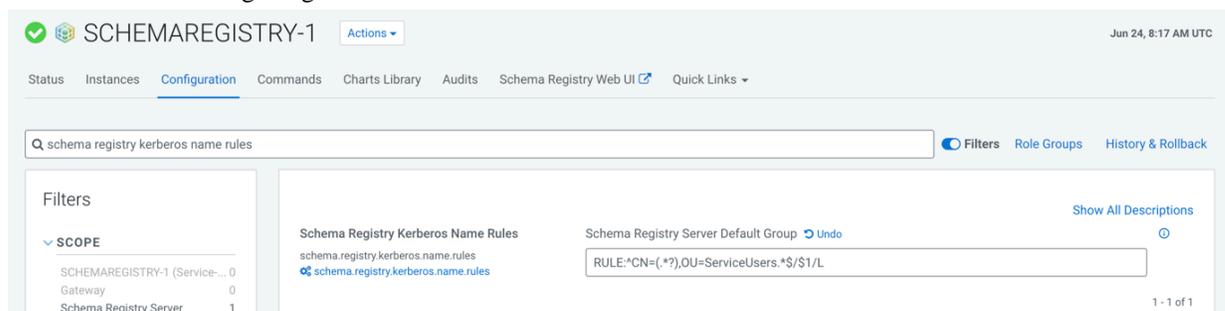


**6.** Click Save Changes.

## Related Information
Configure TLS Encryption Manually for Schema Registry

# Schema Registry authorization through Ranger access policies

User and group access to various Schema Registry functions is controlled through Apache Ranger.

Predefined access policies for Schema Registry allow the administrator to quickly add a user or user group to specify:

- Who can add or evolve schemas to a schema metadata.
- Who can view and edit schemas within a schema metadata.
- Who can upload the SerDes JAR files.

If a higher level of granularity is necessary, the administrator can create an access policy and add the user or user group to this custom policy.

**Related Information**

Predefined access policies for Schema Registry

Adding the user or group to a predefined access policy

Creating a custom access policy

## Predefined access policies for Schema Registry

Based on a user's responsibilities, you can add users or user groups to one or more of the predefined access policies for Schema Registry and you can specify if they have the permission to create, read, update, or delete access policies..

The following image shows the predefined access policies for Schema Registry:



The following table describes the predefined access policies for Schema Registry:

| Access Policy | Description |
|---|---|
| all - export-import | Allows users to import and export schemas to or from the Schema Registry service. For example, a user can import a JSON file with schemas from a Confluent Kafka topic to Cloudera Schema Registry. |
| all - serde | Allows users to store metadata regarding the format of how data should be read and how it should be written. Users can store JAR files for serializers and deserializers and then map the SerDes to the schema. |

| Access Policy | Description |
|---|---|
| all - schema-group, schema-metadata | Allows users to access the schema groups and schema metadata. |
| all - schema-group, schema-metadata, schema-branch | Allows users to access the schema groups, schema metadata, and schema branch. |
| all - registry-service | Allows users to access the Schema Registry service. If a user is added to this policy, the user can access all Schema Registry entities. |
| all - schema-group, schema-metadata, schema-branch, schema-version | Allows users to access the schema groups, schema metadata, schema branch, and schema version. |

**Related Information**

Schema Registry authorization through Ranger access policies

Adding the user or group to a predefined access policy

Creating a custom access policy

# Adding the user or group to a predefined access policy

When an authenticated user attempts to view, create, edit, or delete a Schema Registry entity, the system checks whether the user has privileges to perform that action. These privileges are determined by the Ranger access policies that a user is associated with.

**Before you begin**

For Ranger policies to work, you must have a user group named schemaregistry. If you use UNIX PAM, the schemaregistry user group must be on the node that hosts Schema Registry.

**About this task**

Determine the permissions required by a user or user group, and accordingly add the user or group to the appropriate predefined access policy.

Each predefined access policy controls access to one or more Schema Registry entities.

**Procedure**

1. From the Cloudera Manager home page, click the Ranger link.
   The **Ranger** management page appears.

**2.** Click Ranger Admin Web UI.



The **Ranger Log In** page appears.

**3.** Enter your user name and password to log in.
The **Ranger Service Manager** page appears.

The page is organized by service. Each cluster is listed under its respective service. For example, the Schema Registry clusters in the environment are listed under Schema Registry.

**4.** Select a cluster from the Schema Registry section.
The **List of Policies** page appears.



**5.** Click the ID of a policy.
The **Edit Policy** page appears.

6. In the Allow Conditions section, add the user or group to the respective Select User or Select Group field.



7. In the Policy Conditions field, enter the appropriate IP address.

8. From the Permissions field, select the appropriate permission.

9. Click Save.

### Results

The user now has the rights according to the policy and the permissions you assigned to the user. These rights apply to all objects in the entities unless you specified otherwise in the Policy Conditions field.

### Related Information

Schema Registry authorization through Ranger access policies

Predefined access policies for Schema Registry

Creating a custom access policy

# Creating a custom access policy

You can create a custom access policy for a specific Schema Registry entity, specify an access type, and add a user or user group to the policy.

### Before you begin

Determine and note down the following information:

- The schema registry entity that the user needs access to.
- Whether the user requires all objects in the entity or specific objects.
- Whether the user needs read, view, edit, or delete permissions to the entity.
- If there are any IP addresses to include or exclude from the user's access.

### About this task

With a custom policy you can specify the Schema Registry entity and the type of access the user requires.

### Procedure

1. Go to the **Ranger List of Policies** page.

**2.** Click Add New Policy.



The **Create Policy** page appears.

**3.** Enter a unique name for the policy.

**4.** Optionally, enter a keyword in the Policy Label field to aid in searching for a policy.

**5.** Select a Schema Registry entity. You can choose the Schema Registry service, schema group, or SerDe. Then, perform one of the following tasks:

- If you want the user to access all the objects in the entity, enter *.
- If you want to specify the objects in the entity that a user can access, enter the name of the object in the text field.

**6.** Optionally, enter a description.

**7.** In the Allow Conditions section, add the user or group to the respective Select User or Select Group field.



**8.** Optionally, from the Policy Conditions field, enter the appropriate IP address.

**9.** From the Permissions field, select the appropriate permission.

**10.** Click Save.

**Results**

The user now has the rights according to the policy and the permissions you assigned to the user.

**Related Information**

Schema Registry authorization through Ranger access policies

Predefined access policies for Schema Registry

Adding the user or group to a predefined access policy

# Schema Registry authentication through OAuth2 JWT tokens

You can use OAuth2 JSON Web Token (JWT) in Schema Registry for authentication. Authorization continues to be implemented in Ranger; however, you can obtain the principal from a JWT token.

The flow for authenticating with OAuth2 tokens is as follows:

1. A client requests a token from the OAuth2 service.

   During Schema Registry startup the application obtains the public keys needed for validating the incoming tokens.
2. The client sends the HTTP requests to Schema Registry and these requests contain the bearer token in the HTTP header.
3. Schema Registry validates the token.

The following image shows the authentication flow with OAuth2 tokens:
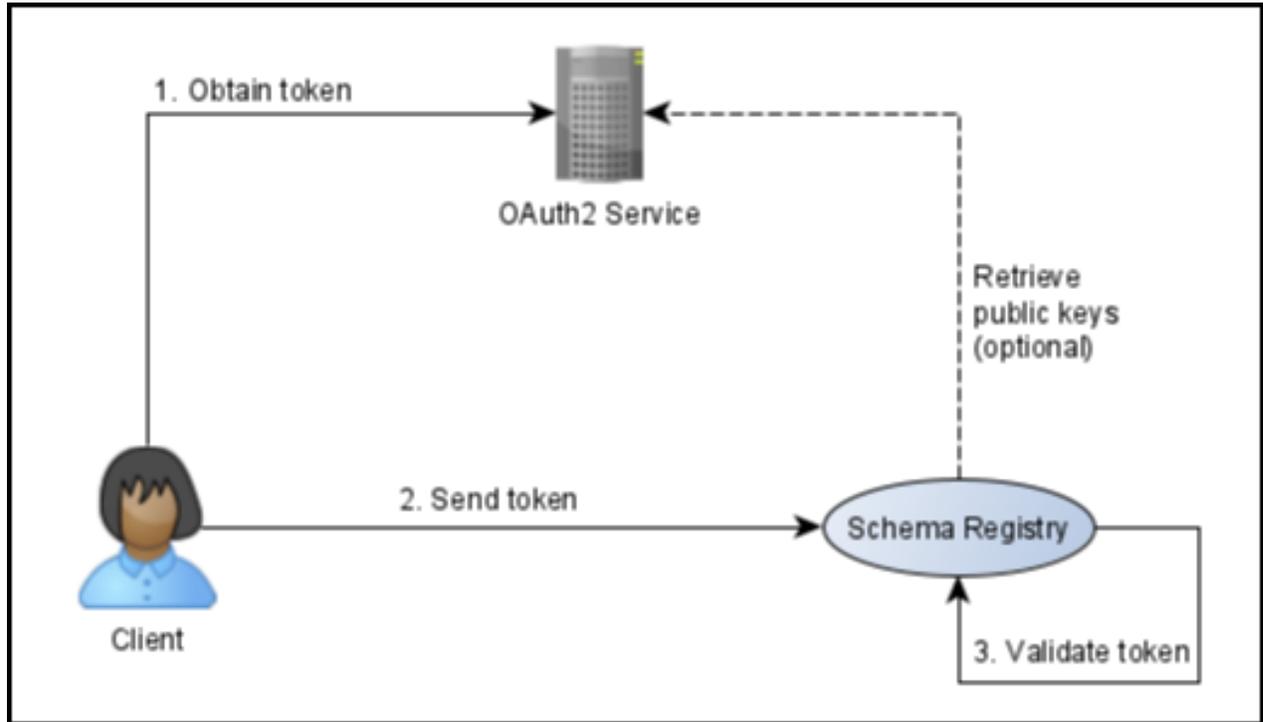


**Note:** Schema Registry currently supports JWT tokens only. There is no support for opaque tokens. The following is an example of a JWT token:

```
{
  "kid": "3",
  "alg": "HS256"
}
--------------------------------------------------
{
  "iss": "sender",
  "aud": "receiver",
  "exp": 1644492815,
  "jti": "5vgglGQCjC9_WZJMJg7mHQ",
  "iat": 1644492515,
  "sub": "abigel"
}
--------------------------------------------------
<signature>
```

The flow for authorization is as follows:

1. Once the token is validated, the principal is extracted from the JWT token. By default, the principal is stored in the sub field.
2. The principal is passed to Ranger which performs the authorization.

# JWT algorithms

Similarly to Kafka, Schema Registry also uses Jose4J for validating the JWT tokens and their signatures.

This library supports a range of signing algorithms: HS256, HS384, HS512, RS256, RS384, and RS512.

For more information, see *Bitbucket jose4j Wiki*.

**Related Information**
Bitbucket jose4j Wiki

# Public key and secret storage

Learn about public key, private key, and secret in JSON Web Token (JWT). Also learn about JSON Web Key (JWK), keystore, and property that Schema Registry supports for storing the public key or the secret.

When JWTs are signed with RSA, there is a private and public key pair. The private key is located on the OAuth2 server and is hidden from you. Schema Registry uses the public key for validating the signature of the JWT token.

When JWTs are signed with HMAC, there is a secret which is shared by all parties. The secret is used for signing the token and also for verifying it.

Schema Registry supports the following ways to store the public key or the secret:

* JWK

  JSON Web Key is a data structure that describes a key. When you have multiple keys collected in a set, that data structure is named JWKS. A JWKS contains a collection of keys.

  Usually, there is a public web service that exposes the JWKS. You can obtain the JWKS through an HTTP request. Other transportation methods are possible, for example, the keys can be stored in a file or on a network storage.

  The keys are usually short lived (depending on the provider the validity period ranges from one day to one week). For this reason, Schema Registry runs a thread every 5 minutes to refresh the keys. The interval can be customized.
* Keystore

  The keys can be stored in a Java keystore file. You need to ensure that Schema Registry has access to the file and permission to read the key.
* Property

  The public key or secret can be stored directly in Schema Registry. In this case, you enter the key in Cloudera Manager and Schema Registry loads it during startup. This option is useful when the public key expires rarely and you do not want to depend on an external JWK service for managing the keys.

# Authentication using OAuth2 with Kerberos

It is possible to have both Kerberos and OAuth2 enabled at the same time for Schema Registry.

OAuth2 is added as yet another authentication layer to Schema Registry. It is possible to have both Kerberos and OAuth2 enabled at the same time. In this scenario, if either one of them succeeds in authenticating the client, the client is given a pass.

This setup can be useful for cases when you have different services communicating with Schema Registry and some of them use Kerberos while others rely on OAuth2.

# Schema Registry server configuration

Learn how to configure general settings for Schema Registry server. Also learn about the extra parameters which you can set when storage type is JWK, keystore, or property.

## General settings

| Property | Data type | Description |
|---|---|---|
| schema.registry.oauth.enabled | Boolean | Select this option to enable OAuth2 authentication. |
| schema.registry.oauth.key.store.type | Enum | Select the type of the key storage where the public key is read from. Possible values are: property, keystore, jwk. Depending on the chosen value, additional configuration might be necessary. These are detailed in the following sections. |
| schema.registry.oauth.jwt.principal.claim.name | String | The JWT token needs to contain the principal which is used during Ranger authorization. By default, it is assumed that the sub claim contains the principal, but this can be modified with this parameter. |
| schema.registry.oauth.jwt.expected.audience | String | The JWT token can optionally contain an audience aud claim. When this claim is present, the same audience value needs to be expected on the server side, otherwise the token is considered invalid. |
| schema.registry.oauth.jwt.expected.issuer | String | The JWT token can optionally contain an issuer iss claim. You can configure Schema Registry to only accept tokens issued by a specific issuer. |
| schema.registry.oauth.clock.skew | Integer | The clock of the server issuing the token might not be in sync with the clock where Schema Registry is running. You can adjust this value to tolerate a certain difference between the two clocks (in seconds). |

## JWK configuration settings

When storage type is JSON Web Key (JWK), you can also apply the following parameters.

| Property | Data type | Description |
|---|---|---|
| schema.registry.oauth.jwks.url | String | URL to the server issuing the JWK keys. This can also be a local file if the URL starts with file://. |
| schema.registry.oauth.jwks.refresh.ms | Long | Refresh interval for reading the keys from the JWK server. Default value is 30000 ms (30 seconds). |

The following parameters are optional. When the keys are downloaded from a remote server, you might need special configuration for accessing the server.

| Property | Data type | Description |
|---|---|---|
| schema.registry.oauth.jwks.httpClient.basic.user | String | If the JWK server requires basic authentication, then you can provide the username. |

| Property | Data type | Description |
| --- | --- | --- |
| schema.registry.oauth.jwks.httpClient.basic.password | String | If the JWK server requires basic authentication, then you can provide the password. |
| schema.registry.oauth.jwks.httpClient.keyStorePath | String | If a key is required for accessing the JWK server, then you can provide the keystore path. |
| schema.registry.oauth.jwks.httpClient.keyStoreType | String | Schema Registry keystore type of HTTP client used for JWK OAuth2. This can be required when keystore type is jwk. |
| schema.registry.oauth.jwks.httpClient.keyPassword | String | Schema Registry key password of HTTP client used for JWK OAuth2. This can be required when keystore type is jwk. |
| schema.registry.oauth.jwks.httpClient.keyStorePassword | String | Schema Registry keystore password of HTTP client used for JWK OAuth2. This can be required when keystore type is jwk. |
| schema.registry.oauth.jwks.httpClient.keyStoreProvider | String | Schema Registry keystore provider of HTTP client used for JWK OAuth2. This can be required when keystore type is jwk. |
| schema.registry.oauth.jwks.httpClient.keyManagerFactoryAlgorithm | String | Schema Registry algorithm of KeyManagerFactory for HTTP client used for JWK OAuth2. This can be required when keystore type is jwk. |
| schema.registry.oauth.jwks.httpClient.keyManagerFactoryProvider | String | Schema Registry KeyManagerFactory provider for HTTP client used for JWK OAuth2. This can be required when keystore type is jwk. |
| schema.registry.oauth.jwks.httpClient.trustStorePath | String | You can add the certificate of the JWK server to a truststore. |
| schema.registry.oauth.jwks.httpClient.trustStoreType | String | Schema Registry truststore type of HTTP client used for JWK OAuth2. This can be required when keystore type is jwk. |
| schema.registry.oauth.jwks.httpClient.trustStorePassword | String | Schema Registry truststore password of HTTP client used for JWK OAuth2. This can be required when keystore type is jwk. |
| schema.registry.oauth.jwks.httpClient.trustStoreProvider | String | Schema Registry truststore provider of HTTP client used for JWK OAuth2. This can be required when keystore type is jwk. |
| schema.registry.oauth.jwks.httpClient.trustManagerFactoryAlgorithm | String | Schema Registry TrustManagerFactory algorithm for HTTP client used for JWK OAuth2. This can be required when keystore type is jwk. |
| schema.registry.oauth.jwks.httpClient.trustManagerFactoryProvider | String | Schema Registry TrustManagerFactory provider for HTTP client used for JWK OAuth2. This can be required when keystore type is jwk. |
| schema.registry.oauth.jwks.httpClient.protocol | String | HTTPS security protocol. By default, it is TLS. |

### Keystore configuration settings

When storage type is keystore, you can also apply the following parameters.

| Property | Data type | Description |
| --- | --- | --- |
| schema.registry.oauth.keystore.public.key.keystore | String | Path to the keystore file. Ensure the file is readable by Schema Registry. |
| schema.registry.oauth.keystore.public.key.keystore.alias | String | The alias of the key within the keystore. |

| Property | Data type | Description |
| --- | --- | --- |
| schema.registry.oauth.keystore.public.key.keystore.password | String | Password for reading the keystore. |

### Property configuration settings

When storage type is property, you can also apply the following parameters.

| Property | Data type | Description |
| --- | --- | --- |
| schema.registry.oauth.property.public.key.property | String | The public key or the secret. |
| schema.registry.oauth.property.key.algorithm | Enum | The algorithm of the key. The values are: RS256, HS256. |

## Configuring the Schema Registry client

Learn how to configure the Schema Registry client to access the server.

When running together with Kafka, the existing parameters still apply.

The client first sends a request to the OAuth2 auth server and requests a token. Configure the settings required to access the server.

| Property | Data type | Description |
| --- | --- | --- |
| schema.registry.auth.type | String | It needs to be set to oauth2. |
| schema.registry.oauth.server.url | String | URL of the server issuing the tokens. |
| schema.registry.oauth.client.id | String | ID of the client. |
| schema.registry.oauth.secret | String | Secret. |
| schema.registry.oauth.scope | String | Scope (optional). |
| schema.registry.oauth.request.method | String | HTTP request method. By default, it is post. |

# Configuring custom Kerberos principal for Schema Registry

The Kerberos principal for Schema Registry is configured by default to use the same service principal as the default process user. However, you can change the default setting by providing a custom principal in Cloudera Manager.

### Procedure

1. Go to your cluster in Cloudera Manager.
2. Select Schema Registry from the list of services.
3. Go to the **Configuration** tab.
4. Search for the Kerberos Principal by entering kerberos in the search field.
5. Enter a custom name in the Kerberos Principal field.
6. Click Save changes.
7. Click  Action Restart  next to the Schema Registry service name to restart the service.