Cloudera Manager 7.13.2

Encrypting Data in Transit in Cloudera Manager

Date published: 2020-11-30
Date modified: 2026-02-27

CLOUDERA

https://docs.cloudera.com/

Legal Notice

Contents

Encrypting Data in Transit

How to configure TLS/SSL encryption in Cloudera Manager.

Transport Layer Security (TLS) 1.2 is an industry standard set of cryptographic protocols for securing communications over a network. TLS evolved from Secure Sockets Layer (SSL). Because the SSL terminology is still widely used, Cloudera software and documentation refer to TLS as TLS/SSL, but the actual protocol used is TLS. SSL is not used in Cloudera software.

In addition to TLS/SSL encryption, HDFS and HBase transfer data using remote procedure calls (RPCs). To secure this transfer, you must enable RPC encryption.

For instructions on enabling TLS/SSL and RPC encryption, see the following topics:

TLS/SSL and Its Use of Certificates

TLS/SSL provides privacy and data integrity between applications communicating over a network by encrypting the packets transmitted between endpoints (ports on a host, for example). Configuring TLS/SSL for any system typically involves creating a private key and public key for use by server and client processes to negotiate an encrypted connection at runtime. In addition, TLS/SSL can use certificates to verify the trustworthiness of keys presented during the negotiation to prevent spoofing and mitigate other potential security issues.

Setting up Cloudera clusters to use TLS/SSL requires creating private key, public key, and storing these securely in a keystore, among other tasks. Although adding a certificate to the keystore may be the last task in the process, the lead time required to obtain a certificate depends on the type of certificate you plan to use for the cluster.

Certificates Overview

A certificate is digitally signed, typically by a certificate authority (CA) that indirectly (through a chain of trust) verifies the authenticity of the public key presented during the negotiation. Certificates can be signed in one of the three different ways shown in the table:

Type	Usage Note
Public CA-signed certificates	Recommended. This type of certificate is signed by a public certificate authority (CA), such as Symantec or Comodo. Public CAs are trusted third-parties whose certificates can be verified through publicly accessible chains of trust. Using this type of certificate can simplify deployment because security infrastructure, such as root CAs, are already contained in the Java JDK and its default truststore.
Internal CA-signed certificates	This type of certificate is signed by your organization's internal CA. Organizations using OpenSSL Certificate Authority, Microsoft Active Directory Certificate Service, or another internal CA system can use this type of certificate.
Self-signed certificates	Not recommended for production deployments. Self-signed certificates are acceptable for use in non-production deployments, such as for proof-of-concept setups.

During the process of configuring TLS/SSL for the cluster, you typically obtain a certificate for each host in the cluster, and re-use the certificate obtained in a given format (JKS, PEM) as needed for the various services (daemon roles) supported by the host. For information about converting formats, see “How to Convert File Encodings (DER, JKS, PEM) for TLS/SSL Certificates and Keys”. As an alternative to creating discrete certificates for each host in the cluster, Cloudera cluster components support wildcard domains and SubjectAlternateName certificates.

Wildcard Domain Certificates and SAN Certificates Support

Cloudera Manager and Cloudera support the use of wildcard domain certificates and SAN certificates.

A wildcard certificate—a certificate with the common name *, as in *.example.com, rather than a specific host name—can be used for any number of first level sub-domains within a single domain name. For example, a wildcard certificate can be used with host-1.example.com, host-2.example, host-3.example.com, and so on.

Certificates obtained from public CAs are not free, so using wildcard certificates can reduce costs. Using wildcard certificates also makes it easier to enable encryption for transient clusters and for clusters that need to expand and shrink, since the same certificate and keystore can be re-used.

Wildcard Certificates	Wildcard certificates can be used by all hosts within a given domain. Using wildcard certificates for all hosts in the cluster can reduce costs but also exposes greater potential risk.
SubjectAlternativeName Certificates	SubjectAlternativeName (SAN) certificates are bound to a set of specific DNS names. A single SAN certificate can be used for all hosts or a subset of hosts in the cluster. SAN certificates are used in Cloudera Manager high-availability (HA) configurations.

Renew Certificates Before Expiration Dates

The signed certificates you obtain from a public CA (or those you obtain from an internal CA) have an expiration date, such as that shown in this excerpt:

```
$ openssl x509 -in cacert.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 11485830970703032316 (0x9f65de69ceef2ffc)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=MD, L=Baltimore, CN=Test CA/emailAddress=test@example.com
        Validity
            Not Before: Jan 24 14:24:11 2017 GMT
            Not After : Feb 23 14:24:11 2018 GMT
        Subject: C=US, ST=MD, L=Baltimore, CN=Test CA/emailAddress=test@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:b1:7f:29:be:78:02:b8:56:54:2d:2c:ec:ff:6d:
                    ...
                    39:f9:1e:52:cb:8e:bf:8b:9e:a6:93:e1:22:09:8b:
```

Expired certificates cause most cluster operations to fail. Cloudera Manager Agent hosts, for example, will not be able to validate the Cloudera Manager Server host and will fail to launch the cluster nodes. Administrators should note expiration dates of all certificates when they deploy the certificates to the cluster nodes and setup reminders to allow enough time to renew.

Tip: Use OpenSSL to check the expiration dates for certificates already deployed:

```
openssl x509 -enddate -noout -in /OPT/CLOUDERA/SECURITY/PKI/$(HOSTNAME
 -F)-SERVER.CERT.PEM
```

Related Information

Transport Layer Security (TLS) 1.2

How to Convert File Encodings (DER, JKS, PEM) for TLS/SSL Certificates and Keys

Understanding Keystores and Truststores

Configuring Cloudera Manager Server and cluster components to use TLS/SSL requires obtaining keys, certificates, and related security artifacts.

Java Keystore and Truststore

All clients in a Cloudera Manager cluster configured for TLS/SSL need access to the truststore to validate certificates presented during TLS/SSL session negotiation. The certificates assure the client or server process that the issuing authority for the certificate is part of a legitimate chain of trust.

The standard Oracle Java JDK distribution includes a default truststore (cacerts) that contains root certificates for many well-known CAs, including Symantec. Rather than using the default truststore, Cloudera recommends using the alternative truststore, jssecacerts. The alternative truststore is created by copying cacerts to that filename (jssecacerts). Certificates can be added to this truststore when needed for additional roles or services. This alternative truststore is loaded by Hadoop daemons at startup.

Important: For use with Cloudera clusters, the alternative trust store—jssecacerts—must start as a copy of cacerts because cacerts contains all available default certificates needed to establish the chain of trust during the TLS/SSL handshake. After jssecacerts has been created, new public and private root CAs are added to it for use by the cluster. See “Manually Configuring TLS Encryption for Cloudera Manager”>“On Each Cluster Host” for details.

The private keys are maintained in the keystore.

Note: For detailed information about the Java keystore and truststore, see Oracle documentation:

- Keytool—Key and Certificate Management Tool
- JSSE Reference Guide for Java

Although the keystore and truststore in some environments may comprise the same file, as configured for Cloudera Manager Server and Cloudera clusters, the keystore and truststore are distinct files. For Cloudera Manager Server clusters, each host should have its own keystore, even if the content is identical while using wildcard certificates. Also, each host should have a truststore file, even if the content of the truststore is identical across hosts. This table summarizes the general differences between keystore and the truststore in Cloudera Manager Server clusters.

Keystore	Truststore
Used by the server side of a TLS/SSL client-server connection.	Used by the client side of a TLS/SSL client-server connection.
Typically contains 1 private key for the host system.	Contains no keys of any kind.
Contains the certificate for the host's private key.	Contains root certificates for well-known public certificate authorities. May contain certificates for intermediary certificate authorities.
Password protected. Use the same password for the key and its keystore.	Password-protection not needed. However, if password has been used for the truststore, never use the same password as used for a key and keystore.
Password stored in a plaintext file read permissions granted to a specific group only (OS filesystem permissions set to 0440, hadoop:hadoop).	Password (if there is one for the truststore) stored in a plaintext file readable by all (OS filesystem permissions set to 0440).
No default. Provide a keystore name and password when you create the private key and CSR for any host system.	For Java JDK, cacerts is the default unless the alternative default jsse cacerts is available.
Must be owned by hadoop user and group so that HDFS, MapReduce, YARN can access the private key.	HDFS, MapReduce, and YARN need client access to truststore.

The details in the table above are specific to the Java KeyStore (JKS) format, which is used by Java-based cluster services such as Cloudera Manager Server, Cloudera Management Service, and many (but not all) Cloudera components and services. See “Certificate Formats (JKS, PEM) and Cluster Components” for information about certificate and key file type used various processes.

Cloudera Services as TLS/SSL Servers and Clients

Cluster services function as a TLS/SSL server, client, or both:

Component	Client	Server
HBase	⊘	✓
HDFS	✓	✓
Hive	✓	✓
Cloudera Data Explorer (Hue) (Data Explorer is a TLS/SSL client of HDFS, MapReduce, YARN, HBase, and Oozie.)	✓	⊘
MapReduce	✓	✓
Oozie	⊘	✓
YARN	✓	✓
ZooKeeper	✓	✓

Daemons that function as TLS/SSL servers load the keystores when starting up. When a client connects to an TLS/SSL server daemon, the server transmits the certificate loaded at startup time to the client, and the client then uses its truststore to validate the certificate presented by the server.

Certificate Formats (JKS, PEM) and Cluster Components

Cloudera Manager Server, Cloudera Management Service, and many other Cloudera services use JKS formatted keystores and certificates. Cloudera Manager Agent, Data Explorer, Impala, and other Python or C++ based services require PEM formatted certificates and keystores rather than Java. Specifically, PEM certificates conform to PKCS #8, which requires individual Base64-encoded text files for certificate and password-protected private key file. The table summarizes certificate types required by several components.

Component	JKS	PEM
HBase	✓	⊘
HDFS	✓	⊘
Hive (Hive clients and HiveServer 2)	✓	⊘

Component	JKS	PEM
Cloudera Data Explorer (Hue)		
Impala		
MapReduce		
Oozie		
Solr		
YARN		
ZooKeeper		

For more information, see:

- “How to Convert File Encodings (DER, JKS, PEM) for TLS/SSL Certificates and Keys”
- OpenSSL Cryptography and TLS/SSL Toolkit

Recommended Keystore and Truststore Configuration

Cloudera recommends the following for keystores and truststores for Cloudera Manager clusters:

- Create a separate keystore for each host. Each keystore should have a name that helps identify it as to the type of host—server or agent, for example. The keystore contains the private key and should be password protected.
- Create a single truststore that can be used by the entire cluster. This truststore contains the root CA and intermediate CAs used to authenticate certificates presented during TLS/SSL handshake. The truststore does not need to be password protected.

The steps included in “Manually Configuring TLS Encryption for Cloudera Manager”>“On Each Cluster Host” follow this approach.

Related Information

How to Convert File Encodings (DER, JKS, PEM) for TLS/SSL Certificates and Keys

Manually Configuring TLS Encryption for Cloudera Manager

keytool - Key and Certificate Management Tool

Java Secure Socket Extension (JSSE) Reference Guide

Choosing manual TLS or Auto-TLS

An explanation of the difference between manual TLS and Auto-TLS in Cloudera Manager.

Wire encryption protects data in motion, and Transport Layer Security (TLS) is the most widely used security protocol for wire encryption. TLS provides authentication, privacy and data integrity between applications communicating over a network by encrypting the packets transmitted between endpoints. Users interact with Hadoop clusters via browser or command line tools, while applications use REST APIs or Thrift.

Overview of enabling TLS manually

The typical process to enable wire encryption on Cloudera on premises clusters is described below.

Get Certificates

- Generate a public/private keypair on each host
- Generate the Certificate signing request (CSR) for all the hosts.
- Get the CSR signed by the company's internal Certificate Authority (CA).
- Generate keystore & truststore and deploy them across all the cluster hosts.

Cluster configuration

- For each service, enable TLS by setting the keystore and truststore configuration.
- Restart the affected components before proceeding to enable TLS for the next service.
- Make the required changes outside of the cluster manager's UI (like setting up truststore, Enabling Knox SSL, etc.)

Ongoing Maintenance

- For new service installation, the keystore and truststore information need to be configured for the service. Restart the impacted services.
- For each new host to be added to the cluster, admins would have to perform the steps from the "Get Certificates" section (only for the new hosts).
- The certificates are rotated before they expire.

Auto-TLS feature in Cloudera Manager

The process described above can be a significant effort in large deployments, often leading to long deployment times and operational difficulties. The Auto-TLS feature automates all the steps required to enable TLS encryption at a cluster level. Using Auto-TLS, you can let Cloudera manage the Certificate Authority (CA) for all the certificates in the cluster or use the company's existing CA. In most cases, all the necessary steps can be enabled easily via the Cloudera Manager UI. This feature automates the following processes:

When Cloudera Manager is used as a Certificate Authority:

- Creates the root Certificate Authority or a Certificate Signing Request (CSR) for creating an intermediate Certificate Authority to be signed by company's existing Certificate Authority (CA)
- Generates the CSRs for hosts and signs them automatically

The following steps are always performed:

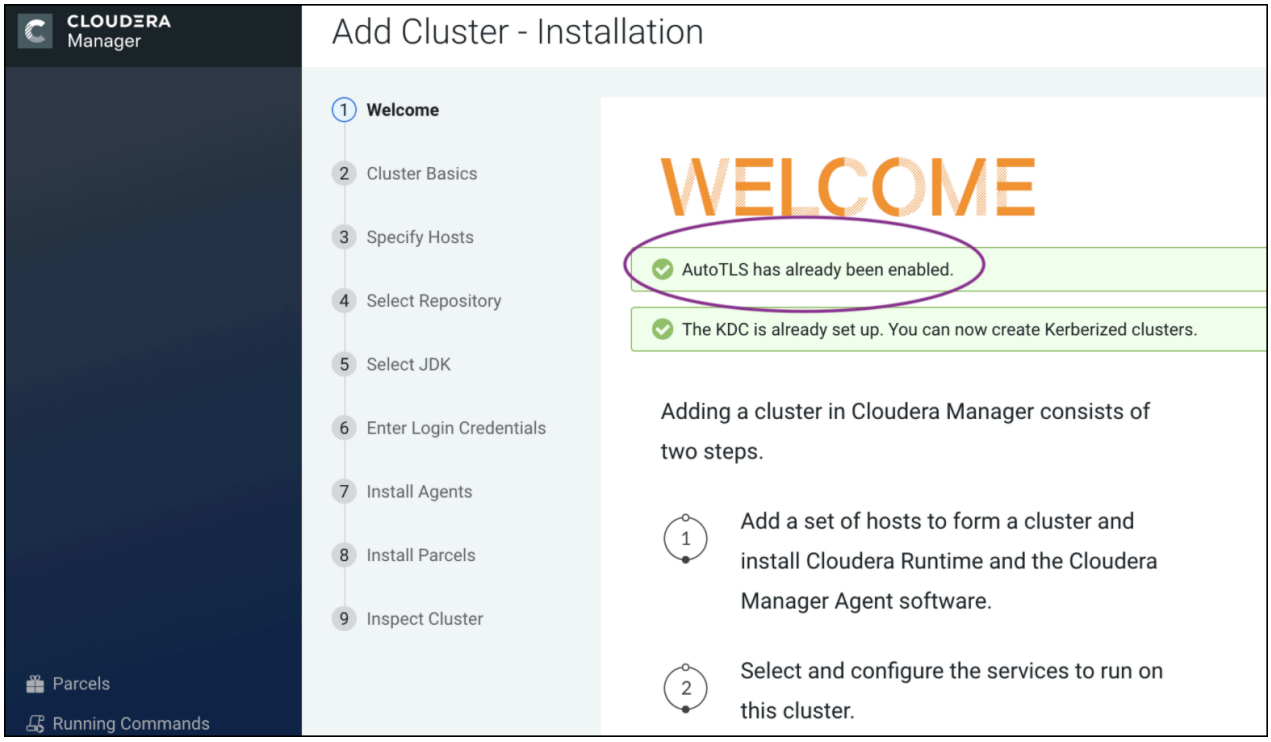
- Creates a keystore and truststore for hosts.
- Deploys the certificates, keystore and truststore to all the hosts in the cluster.
- All the cluster services are then automatically TLS enabled by configuring the keystore and truststore information from a role instance specific directory.
- Enables TLS for Cloudera Manager server and agents.
- After this initial setup, any new service, hosts (or) additional compute clusters setup are automatically TLS enabled by default.
- Provides an automation framework for rotating certificates.

Let us review these options with examples below on a Cloudera on premises 7.1 cluster:

- Use case 1: Using Cloudera Manager to generate an internal CA and corresponding certificates
- Use case 2: Enabling Auto-TLS with an existing Root CA
- Use case 3: Enabling Auto-TLS with existing Certificates

New Cluster deployment

With either of these options, you can reuse the existing TLS settings when creating a new cluster on a Cloudera Manager with Auto-TLS enabled. When you launch the wizard to create a new cluster you should see the following message. Now, when you deploy the cluster, all services will be automatically configured with wire encryption.



Summary

The Auto-TLS functionality not only speeds up the initial setup of the wire encryption but also automates future TLS configuration steps for the cluster. The following table summarizes the differences between the options described in this blog.

Steps	HDP/EDH (manual)	Cloudera on premises use case 1 - Using Cloudera Manager to generate an internal CA and corresponding certificates	Cloudera on premises use case 2 - Enabling Auto-TLS with an existing Root CA	Cloudera on premises use case 3 - Enabling Auto-TLS with Existing Certificates
Generate CSR	Manual	Automated	Automated	Manual
CSR Signed by CA	Manual	Automated	One-time	Manual
Deploy certificate to all hosts	Manual	Automated	Automated	Automated
Configuration for each service	Manual	Automated	Automated	Automated
Cluster restarts	Multiple	Once	Once	Once
Configuration steps	Manual	Automated	Automated	Automated
New Service steps	Manual	Automated	Automated	Automated
New Host cert. generation	Manual	Automated	Automated	Manual

The Auto-TLS feature significantly reduces the overhead of TLS management of your cluster, thus providing increased security with reduced operational overhead and helps you stay focused on your customers and their workloads.

Related Information

Use case 1: Use Cloudera Manager to generate internal CA and corresponding certificates
Use case 2: Enabling Auto-TLS with an intermediate CA signed by an existing Root CA
Use case 3: Enabling Auto-TLS with Existing Certificates

SAN Certificates

A SubjectAlternativeName (SAN) certificate is a certificate that uses the SubjectAlternativeName extension to associate the resulting certificate with multiple specific host names. SAN certificates are supported while using Auto-TLS only if custom certificates are generated. SAN certificates are not supported with the internal Cloudera Manager Certificate Authority.

Configuring TLS Encryption for Cloudera Manager Using Auto-TLS

Use Auto-TLS to simplify the process of configuring TLS encryption for Cloudera Manager.

About this task

The Auto-TLS feature automates all the steps required to enable TLS encryption at a cluster level. Using Auto-TLS, you can let Cloudera manage the Certificate Authority (CA) for all the certificates in the cluster or use the company's existing CA. In most cases, all the necessary steps can be enabled easily via the Cloudera Manager UI. This feature automates the following processes:

When Cloudera Manager is used as a Certificate Authority:

- Creates the root Certificate Authority or a Certificate Signing Request (CSR) for creating an intermediate Certificate Authority to be signed by company's existing Certificate Authority (CA)
- Generates the CSRs for hosts and signs them automatically

The following steps are always performed

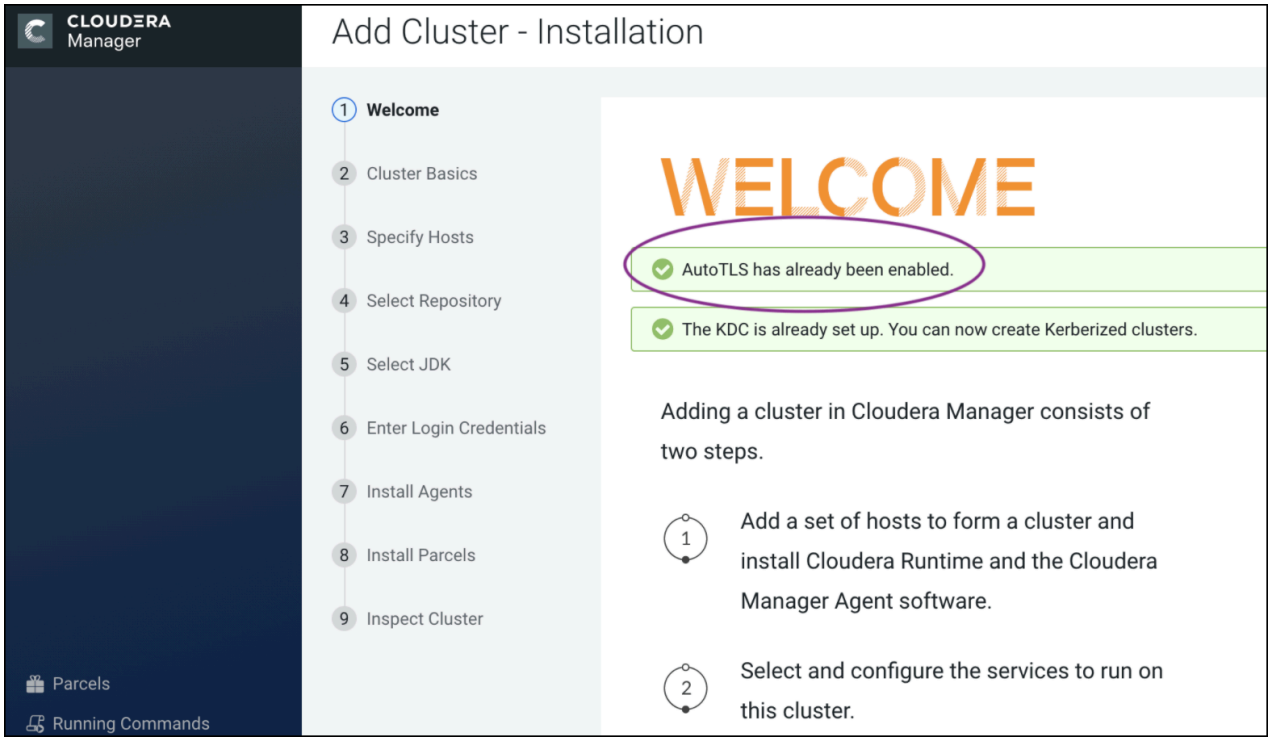
- Creates a keystore and truststore for hosts.
- Deploys the certificates, keystore and truststore to all the hosts in the cluster.
- All the cluster services are then automatically TLS enabled by configuring the keystore and truststore information from a role instance specific directory.
- Enables TLS for Cloudera Manager server and agents.
- After this initial setup, any new service, hosts (or) additional compute clusters setup are automatically TLS enabled by default.
- Provides an automation framework for rotating certificates.

Let us review these options with examples below on a Cloudera Base on premises 7.1 cluster:

- Use case 1: Using Cloudera Manager to generate an internal CA and corresponding certificates
- Use case 2: Enabling Auto-TLS with an existing Root CA
- Use case 3: Enabling Auto-TLS with existing Certificates

New Cluster deployment

With either of these options, you can reuse the existing TLS settings when creating a new cluster on a Cloudera Manager with Auto-TLS enabled. When you launch the wizard to create a new cluster you should see the following message. Now, when you deploy the cluster, all services will be automatically configured with wire encryption.



Summary

The Auto-TLS functionality not only speeds up the initial setup of the wire encryption but also automates future TLS configuration steps for the cluster. The following table summarizes the differences between the available options.

Steps	HDP/EDH (manual)	Cloudera on premises use case 1 - Using Cloudera Manager to generate an internal CA and corresponding certificates	Cloudera on premises use case 2 - Enabling Auto-TLS with an existing Root CA	Cloudera on premises use case 3 - Enabling Auto-TLS with Existing Certificates
Generate CSR	Manual	Automated	Automated	Manual
CSR Signed by CA	Manual	Automated	One-time	Manual
Deploy certificate to all hosts	Manual	Automated	Automated	Automated
Configuration for each service	Manual	Automated	Automated	Automated
Cluster restarts	Multiple	Once	Once	Once
Configuration steps	Manual	Automated	Automated	Automated
New Service steps	Manual	Automated	Automated	Automated
New Host cert. generation	Manual	Automated	Automated	Manual

The Auto-TLS feature significantly reduces the overhead of TLS management of your cluster, thus providing increased security with reduced operational overhead and helps you stay focused on your customers and their workloads.

Related Information

Manually Configuring TLS Encryption for Cloudera Manager

Auto-TLS Requirements and Limitations

Reference information for Auto-TLS requirements, limitations, and component support.

About this task

Note: AutoTLS feature is limited to some components. For more information, see Auto-TLS Requirements and Limitations.

LDAP does not support AutoTLS feature. When you enable AutoTLS, it does not also enable LDAPS (TLS for LDAP). But if you are using use case 2 or 3 (or provide a Trusted CA Certificates Location with the LDAP server's trustedCertEntry) you can use the keystores and truststores created by AutoTLS for use with LDAPS.

AutoTLS truststore and keystore configurations for Edge Flow Manager with LDAPS are as follows:

```
efm.security.ldap.tls.keyStore: /var/lib/cloudera-scm-agent/agent-cert/cm-au
to-host_keystore.jks
efm.security.ldap.tls.keyStorePassword
efm.security.ldap.tls.trustStore: /var/lib/cloudera-scm-agent/agent-cert/cm-
auto-global_truststore.jks
efm.security.ldap.tls.trustStorePassword
```

The keystore password can be found in the /var/lib/cloudera-scm-agent/agent-cert/cm-auto-host_key.pw file. The truststore password can be found by performing the following steps:

1. Navigate to Cloudera Manager.
2. Change the URL to https://<CM Server Host>:7183/api/v45/certs/truststorePassword.

Rotating Auto-TLS Certificate Authority and Host Certificates

Your cluster security requirements may require that you rotate the auto-TLS CA and certificates.

Using an internal CA (Use case 1)

1. Navigate to Administration Security . Click Rotate Auto-TLS Certificates to launch the wizard.
2. Complete the wizard.

Using a custom CA (Use case 3)

Note: This process is to rotate host certificates only. For rotating both host certificates and the CA, see Rotate Auto-TLS Certificate Authority and Host Certificates.

1. Use the `/cm/commands/addCustomCerts` API command to replace the old certificates with new certificates in CMCA directory for each host. You must run this command for each host separately. An example of a curl command to upload the certificates to Cloudera Manager:

```
curl -u <username>:<password> -X POST --header 'Content-Type:
application/json' --header 'Accept: application/json' -d '{
            "location": "/opt/cloudera/AutoTLS",
            "interpretAsFilenames": true,
            "hostCerts": [ {
                "hostname": "ccycloud-10.vcdp71.root.hwx.site",
                "certificate":
"/tmp/auto-tls/certs/ccycloud-10.vcdp71.root.hwx.site.pem",
            "key":
"/tmp/auto-tls/certs/ccycloud-10.vcdp71.root.hwx.site.pem"
            } ]
```

```
}' 'https://ccycloud-7.vcdp71.root.hwx.site:7183/api/v41/cm/commands/addC
ustomCerts'
```

In the example above, the "location" should be omitted if Auto-TLS was enabled or rotated after 7.1, and the file paths should point to files on the Cloudera Manager server host.

2. Use /hosts/{hostId}/commands/generateHostCerts Cloudera Manager API to deploy the new certificates to each host. You must run this command for each host separately. An example curl command :

```
curl -u <username>:<password> -X POST --header 'Content-Type: application/
json' --header
'Accept: application/json' -d '{ "sshPort" : 22, "userName" : "root", "pas
sword" : "cloudera" }'
'https://ccycloud-7.vcdp71.root.hwx.site:7183/api/v41/hosts/250e1bb7-898
7-419c-a53f-c852c275d299/commands/generateHostCerts'
```

where '250e1bb7-8987-419c-a53f-c852c275d299' in the command above is the hostID.

Auto-TLS Agent File Locations

The certificates, keystores, and password files generated by auto-TLS are stored in /var/lib/cloudera-scm-agent/agent-cert on each Cloudera Manager Agent.

About this task

The filenames are as follows:

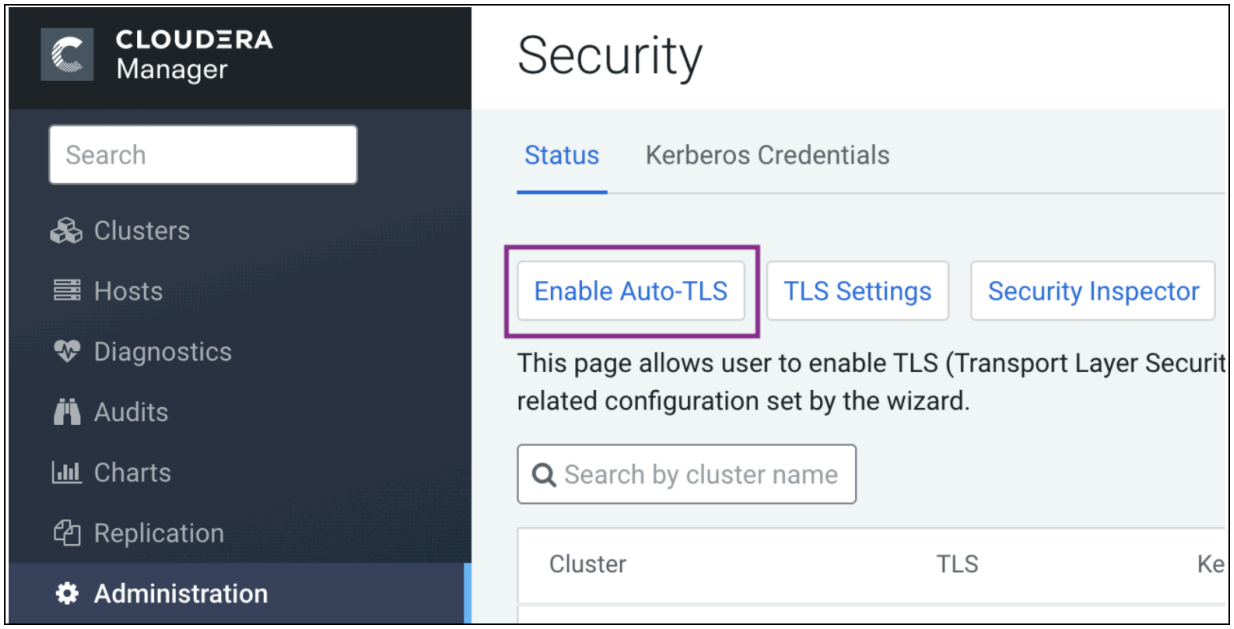
Table 1: Auto-TLS Agent Files

Filename	Description
cm-auto-global_cacerts.pem	CA certificate and other trusted certificates in PEM format
cm-auto-global_truststore.jks	CA certificate and other trusted certificates in JKS format
cm-auto-in_cluster_ca_cert.pem	CA certificate in PEM format
cm-auto-in_cluster_truststore.jks	CA certificate in JKS format
cm-auto-host_key_cert_chain.pem	Agent host certificate and private key in PEM format
cm-auto-host_cert_chain.pem	Agent host certificate in PEM format
cm-auto-host_key.pem	Agent host private key in PEM format
cm-auto-host_keystore.jks	Agent host private key in JKS format
cm-auto-host_key.pw	Agent host private key password file

Use case 1: Use Cloudera Manager to generate internal CA and corresponding certificates

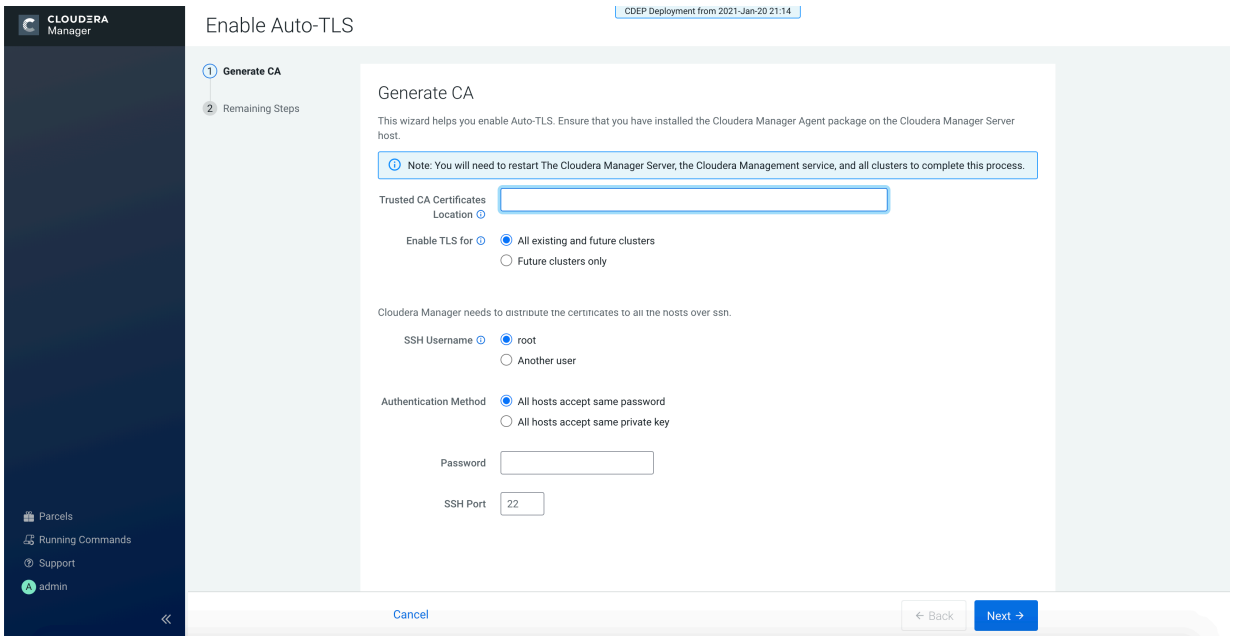
Use Cloudera Manager to create and manage its own Certificate Authority.

1. To choose this option, from Cloudera Manager go to Administration Security (Status tab) Enable Auto-TLS . The Enable Auto-TLS page comes up.



2. The Trusted CA Certificates Location field in the Generate CA section is only for advanced use cases. This field should be left empty unless directed to be filled by Cloudera support personnel or Cloudera professional services. The field should contain a path to a PEM file on the Cloudera Manager host, which contains a list of root CA certificates that should be imported into the truststores of all cluster hosts.

Important: If you are using telemetry publisher, then you should ensure that the Starfield CA cert is part of the trusted CA certs bundle.

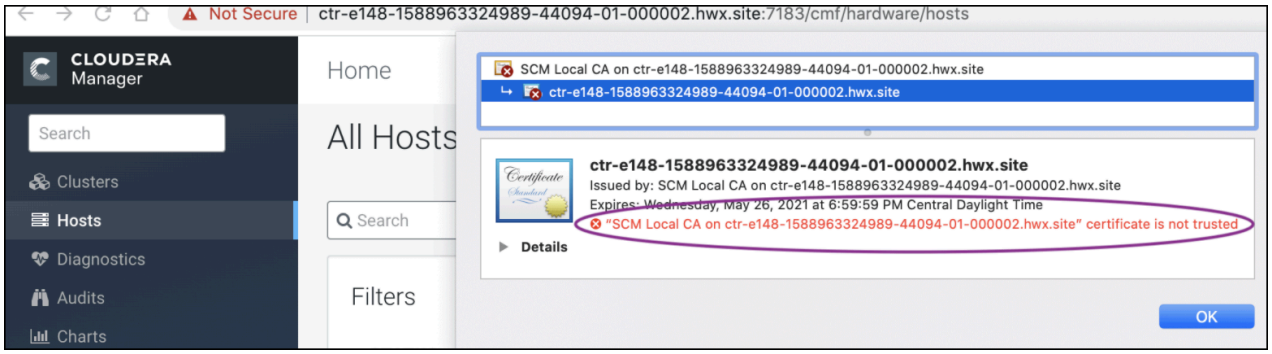


3. From the Enable TLS for: options, select All existing and future clusters to enable Auto-TLS for all existing and future clusters, or select Future clusters only to enable Auto-TLS for future clusters only.
4. Select the required SSH Username option. The available options are root and Another user.

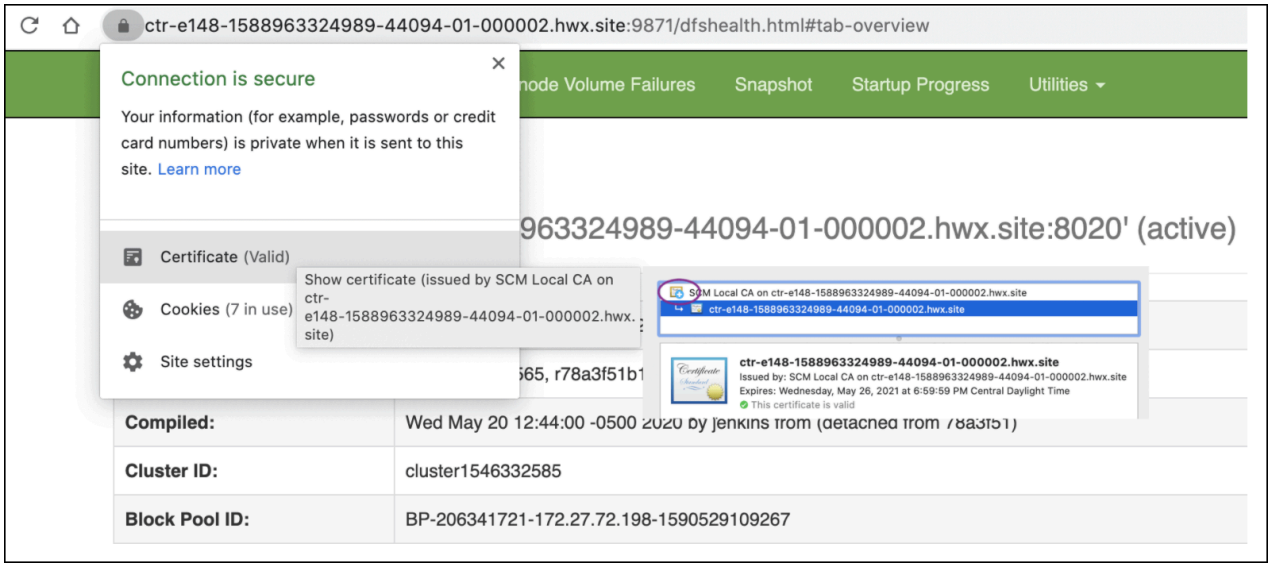
Note: If Another user is selected, ensure that you specify the name of a user having passwordless sudo privileges.

5. Select the required Authentication Method. You can either enable all hosts to accept the same password or you can enable all hosts to accept the same private key.
6. Enter the password in the Password field and verify the SSH Port number.
7. Click Next.

You are prompted to start Cloudera Manager, followed by Cloudera Management Services and any impacted clusters. When you start the Cloudera Manager server, you should see the UI at the TLS port 7183 by default. The browser displays a self-signed certificate from the SCM Local CA authority, as shown below. The browser displays a warning because it is not aware of the Root CA generated by Cloudera Manager. When the Root CA is imported into the client browser's truststore, this warning is not displayed by the browser.



When you set up the cluster, you should see a message stating that Auto-TLS is already enabled. Continue to install the required services. The whole cluster is TLS encrypted. Any new hosts or services are automatically configured. Here is an example of HDFS service with TLS encryption enabled by default (after trusting the root certificate generated by Cloudera Manager).



While this option is the simplest, it may not be suitable for some enterprise deployments where TLS certificates are issued by the company's existing Certificate Authority (CA) to maintain a centralized chain of trust.

Rotate Auto-TLS Certificate Authority and Host Certificates

Your cluster security requirements may require that you rotate the Auto-TLS CA and certificates.

1. Navigate to Administration > Security.
2. Click the Rotate Auto-TLS Certificates button to launch the wizard.
3. Complete the wizard.
4. Restart Cloudera Manager, Cloudera Management Services, and all clusters.

   This process also resets the truststore.

5. If there are other root CA certificates in the truststore, please add them using the Trusted CA Certificates Location field in the Generate CA section.
6. You must restart the Cloudera Management Service and finally restart any clusters that are stale.

Related Information

Use case 2: Enabling Auto-TLS with an intermediate CA signed by an existing Root CA

Use case 3: Enabling Auto-TLS with Existing Certificates

Use case 2: Enabling Auto-TLS with an intermediate CA signed by an existing Root CA

You can make the Cloudera Manager CA become part of an intermediate CA to an existing Root CA.

This is a three-step process. First, make Cloudera Manager generate a Certificate Signing Request (CSR). Second, have the CSR signed by the company's Certificate Authority (CA) and ensure the customer has added that CSR to its intermediate chain as a new certificate authority. Third, get the updated Intermediate CA that now includes the CSR provided by Cloudera from the customer to continue the Auto-TLS setup.

Customizing the certmanager before execution

Cloudera offers the certmanager utility to create the CSR certificate. Basically the certmanager utility triggers an openssl command with a request to generate a key based on the parameters given using a simpler interface.

You can either use the certmanager to generate the CSR or create it manually using openssl command line tools.

While creating a CSR using the certmanager, ensure that you use the correct values. Hence, you need to override some of the defaults the script uses (which can be seen in *Certmanager Options - Using CM's GenerateCMCA API*), as those might work fine in case of Use Case 1 (where everything is self-signed and made by the certmanger), but will not work in case of Use Case 2.

As an example, you can see the default value of cn_name ("SCM Local CA on <host_fqdn>") which can not be used (as cn_name should contain only an FQDN).

One of the most important parameters required is the size of the key. By default (as seen in the above referenced page), certmanager uses a size of 3072 bit, however, in case you need a different size (usually it is 2048 bit) you need to configure it accordingly. For such cases, you prepare a configuration file to customize the certmanager default values.

A local file should be created with the name config.ini on the host from which the certmanager script will be executed. In the following example, you can see the content of such a file that updates the size of the key to 2048 bits, and sets the ca_dn (which is the subject DN added to the CSR) with a value that contains the server name in a DEV environment (hence DC=DEV). Naturally, it can be set according to the customer's needs.

```
[General]
ca_key_args=2048
host_key_args=2048
ca_dn="CN=server_name.example.com,DC=Cloudera,DC=DEV"
```

1. With that file at hand, execute the certmanager script (from the Cloudera Manager node).

   Ensure that the Cloudera Manager is not running, and initialize the script using the below command (replace the XXXXX depending on your openjdk version and assuming the config.ini file you created is in the current folder):

   ```
   export JAVA_HOME=<path-to-java-home>; /opt/cloudera/cm-agent/bin/certman
   ager --location /var/lib/cloudera-scm-server/certmanager setup --config ./
   config.ini --configure-services --stop-at-csr
   ```

   The result of the execution should look like the following:

   ```
   INFO:root:Logging to /var/log/cloudera-scm-agent/certmanager.log Stopping
    after signing the CSR, continue Auto-TLS setup by rerunning ce rtmanager
    setup and passing in --signed-ca-cert <signed_ca_chain.pem>
   ```

2. The CSR (Certificate Signing Request) file will be created under /var/lib/cloudera-scm-server/certmanager/CMCA/private/ca_csr.pem.

   Note: The cloudera-scm user must own the /var/lib/cloudera-scm-server/certmanager/ directory.

   To check the content of the file and validate that it is according to what you configured, you can run the following command:

   ```
   openssl req -in ca_csr.pem -noout -text
   ```

   As this is a request for signing, you use the openssl flag req to see the content of the file. You will see that it was created with 2048 bit, under the protocol of x509v3 which is what is required, and it carries the extension "X509v3 Extended Key Usage: TLS Web Server Authentication, TLS Web Client Authentication".

3. Send that file to the customer. The customer should take that CSR, sign it using its Root Certificate Authority (Root CA), and add the Cloudera Manager CSR as the last part of its Intermediate certificate authority chain.

   By adding the CSR to its Intermediate chain, and returning the Root CA and the updated Intermediate CA to Cloudera, the customer grants Cloudera Manager the ability to be an approved CA, hence Cloudera Manager could create and sign certificates on behalf of the customer to all the cluster nodes.

   Note: The customer, who gave the grant, can revoke it by revoking the CSR from its Intermediate chain. If this is done, all the certificates created by Cloudera Manager using that Intermediate CA will be automatically revoked as well. As the Intermediate and Root CA received are public signed certificates, they can be checked using the command openssl x509 -in <certificate file name> -noout -text. Please execute that command on the Intermediate file you have received and validate that the Cloudera Manager CSR part is in it contains the required extensions X509v3 Basic Constraints: CA: TRUE and X509v3 Key U sage: Key Cert Sign.

4. Proceed with the installation with the following example command (replace JDK version according to the OpenJDK release you have on your platform):

   ```
   export JAVA_HOME=<path-to-java-home>;
   /opt/cloudera/cm-agent/bin/certmanager --location
   /var/lib/cloudera-scm-server/certmanager setup
   --configure-services --trusted-ca-certs /path/to/ca-certs.pem
   --signed-ca-cert /path/to/cm_cert_chain.pem
   X509v3 Basic Constraints: critical CA:TRUE, pathlen:0
   X509v3 Key Usage: critical
   ```

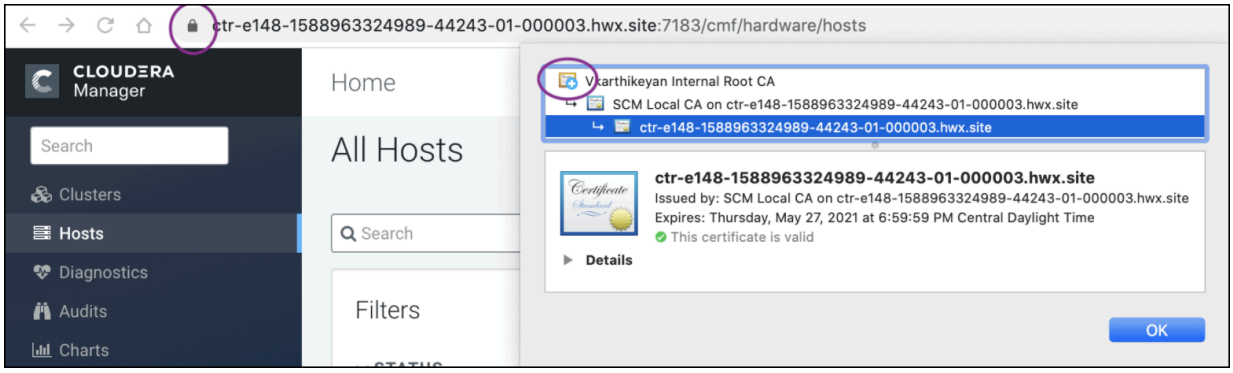
```
Digital Signature, Certificate Sign, CRL Sign
```

Note:

- ca-certs.pem is the Root CA received from the customer in PEM format (text)
- cm_cert_chain.pem should be a chain of the above Root CA + the received Intermediate CA that was sent by the customer to Cloudera already with the Cloudera CSR file embedded inside. To create it, you can run the following, as an example, on the CLI:

```
cat Intermediate.pem ca-certs.pem > cm_cert_chain.pem
```

5. Once the execution is done, start Cloudera Manager on TLS port 7183. If the signed Intermediate certificate is already imported into your browser's truststore, then you should not see any warnings. In the screenshot below, "Vkarthikeyan Internal Root CA" is the root certificate. This certificate is already trusted by the system and has signed by the Cloudera intermediate CA.



Note: In this use case, rotation of the Auto-TLS certificate authority is not supported. Cloudera recommends creating a CSR with a long lifetime to be integrated into the customer Intermediate CA. The host certificates can be rotated by using the generateHostCerts API.

Related Information

Use case 1: Use Cloudera Manager to generate internal CA and corresponding certificates

Use case 3: Enabling Auto-TLS with Existing Certificates

Certmanager Options - Using Cloudera Manager's GenerateCMCA API

Certmanager Options - Using Cloudera Manager's GenerateCMCA API

This article describes how to use the certmanager's GenerateCMCA API.

It generates the CMCA, which is an OpenSSL self-signed cert and CA directory. It creates and signs the certificate for the Cloudera Manager host. Creates an "internal" truststore with the CA certificate only. If trusted certificates were given, loads them into a "global" truststore. certmanager is part of the Cloudera Manager agent package.

Name

certmanager setup- generates CMCA and certificates for the host.

Description

This command initializes the certmanager and setup the certificates for the Cloudera Manager server to run on this host, using those certificates.

The --location option (if any) given to certmanager will decide the location of the directory root where the certmanager will keep its files. This directory will contain sensitive files. It must be backed up and protected accordingly. This directory must not exist prior to calling this command, but must be create-able. (i.e. either its ancestors must exist, or must be create-able).

Options

- --config *CONFIG-FILE-OR-DIR*

  Path for configuration to use. If a directory, read and use all .ini files within it. If a file, must point to a config .ini file.
- --override *SECTION.PROPERTY=VALUE*

  Override config file setting.
- --hostname *DNS-OR-IP*

  Alternate name of local host (for SSL certificates). Only applies if CA type is 'internal'
- --altname *ALT-NAME*

  Alternate name of to add to generated SSL certificates. Multiple names may be supplied by repeating the option. Only applies if CA type is 'internal'.
- --write-cm-init / --skip-cm-init

  Writes a Cloudera Manager init file to set Auto-TLS related parameters. If disabled, only the CA directory will be created, and the init file contents will be printed to stdout.
- --rotate / --no-rotate

  Rotates the CA keys and certificates. If disabled, the command fails if the CA directory already exists.
- --configure-services / --no-configure-services

  Configure new services to use Auto-TLS certificates. If disabled, only agents will use TLS certificates.
- --trusted-ca-certs *TRUSTED_CERTS.PEM*

  Path to trusted CA certs bundle in PEM format. These certs will be imported into the truststore for all hosts.
- --skip-invalid-ca-certs / --fail-invalid-ca-cert

  Whether to skip invalid CA certs in the trusted CA certs bundle. If false, setup will fail if any certs are invalid orduplicates.
- --stop-at-csr / --dont-stop-at-csr

  Whether to stop at signing the CSR. If true, continue the setup by running setup and passing in --signed-ca-cert.
- --signed-ca-cert *SIGNED_CA_CHAIN.PEM*

  Path to signed CA cert chain. Pass this after running setup with the --stop-at-csr option.
- --help

  Show this message and exit.

How to customize CSR fields

Important: If you customize any of the CSR fields by using the “--override” option in an Auto-TLS enabled cluster, then post update, you must restart the Cloudera Manager Server, Cloudera Manager Agents, Cloudera Management Service, and Clusters.

You can customize the CSR fields by using the “--override“ command-line option. The properties available for the --override parameter are mentioned below. An example of how to use the API follows this table.

Property	Default Value	Description
email_address	-	Additional subject alternate names to add to the certioficate.
subject_suffix	-	The subject suffix to be appended to the CN.
ca_key_algo	rsa	CA Key encryption algorithm to be used. Valid values are "rsa", "dsa", "ec".
ca_key_args	3072	Integer value determining the number of bits for the key.
ca_sig_hash_algo	sha256	The hashing algorithm to use when signing.

Property	Default Value	Description
ca_dn	-	The Subject DN to add to the CSR.
ca_expiration	5 years	"YYYMMDD" format date for when certificate expires.Certificate expires at 23:59:59 on the given date at GMT time.
host_expiration	1 year	"YYYMMDD" format date for when host expires.
ca_name	SCM Local CA on <host_fqdn>	Common Name to be used while generating the subject for the certificate.
host_key_algo	rsa	The asymmetric key algorithm to use to generate a CSR for a host.
host_key_args	3072	The parameter to the key algorithm (# bits, curve name, etc.) to generate a CSR for a host.
host_sig_hash_algo	sha256	The hashing algorithm to use in the signature when using the internal CA to sign the given host's CSR.
key_encryption_algo	aes256	The algo to use to encrypt the private key to generate a CSR for a host.

Examples

This example shows how you can use additionalArguments property to pass any override parameters to the certmanager using generateCMCA API.

```
curl -X POST "https://tkarkera-1.tkarkera.root.hwx.site:7183/api/v45/cm/comm
ands/generateCmca" -H "accept: application/json" -H "Content-Type: applicati
on/json" -d
      "{
        "sshPort": 22,
        "userName": "...",
        "password": "...",
        "privateKey": "...",
        "passphrase": "...",
        "location": "/opt/cloudera/CMCA",
        "customCA": false,
        "interpretAsFilenames": true,
        "cmHostCert": "host-cert.pem",
        "cmHostKey": "host-key.pem",
        "caCert": "ca-cert.pem",
        "keystorePasswd": "keystore.pw.txt",
        "truststorePasswd": "truststore.pw.txt",
        "trustedCaCerts": "cacerts.pem",
        "additionalArguments": [
        "--override",
        "ca_expiration=301010",
        "--override",
        "ca_key_args=4096",
        "--override",
        "host_key_args=4096"
        ],
        "hostCerts": [
        {
        "hostname": "...",
        "certificate": "host-cert.pem",
        "key": "host-key.pem",
        "subjectAltNames": [
        "DNS:example.cloudera.com",
        "..."
        ]
```

```
        },
        {
        "hostname": "...",
        "certificate": "...",
        "key": "...",
        "subjectAltNames": [
        "...",
        "DNS:example.cloudera.com"
        ]
        }
        ],
        "configureAllServices": true
        }“
```

Use case 3: Enabling Auto-TLS with Existing Certificates

You can manually generate the certificates signed by an existing Root CA and upload them to Cloudera Manager.

If you have an existing cluster where you need to enable Auto-TLS, or if there is a need to get the host certificates signed individually by the company's existing CA, you can use this option of enabling Auto-TLS with existing certificates. This option adds operational overhead of generating certificates for any new hosts and uploading to Cloudera Manager through an API request. In this option, certificates signed by CA are staged and Auto-TLS is enabled by calling a Cloudera Manager API.

Warning: AutoTLS rejects any certificate chain that is inconsistent with the chain in caCert unless the entire certificate chain is included in the certificate file.

1. Create the Auto-TLS directory /opt/cloudera/AutoTLS in the Cloudera Manager server. The directory must be owned by the cloudera-scm user.
2. Create a public/private key for each host and generate the corresponding Certificate Signing request (CSR). Have these CSRs signed by the company's Certificate Authority (CA). You can generate private keys and CSRs by using your existing PKI tools and processes, or manually with common utilities like keytool or openssl. In this example using openssl, the private key and CSR files are located under the /tmp/auto-tls directory. The password used for the private key is stored in key.pwd.

```
openssl req -newkey rsa:4096 -sha256 -days 365 \
-keyout /tmp/auto-tls/keys/host1.example.com-key.pem \
-out /tmp/auto-tls/host1.example.com.csr \
-passout file:/tmp/auto-tls/keys/key.pwd \
-subj '/CN=host1.example.com, O=Cloudera Test, C=US' \
-extensions san -config <( echo '[req]'; echo 'distinguished_name=req';
 echo 'req_extensions=san';echo '[san]'; echo 'subjectAltName=DNS:host1.
```

```
example.com, DNS:loadbalancer.example.com'; echo 'extendedKeyUsage = ser
verAuth, clientAuth')
```

Important:

- When creating cert for multiple SAN entries, you need to use a comma-separated string.

The same procedure is used for all cluster hosts.

Important:

If the certificate does not have the DNS field, re-submit the CSR to the CA, and request that they generate a certificate that keeps the Subject Alternative Name field intact.

Note that the optional X509v3 Extension Netscape Certificate Type (nsCertType) is deprecated and not recommended. If it is required by your Certificate Authority, please ensure both SSL Server (server) and SSL Client (client) types are set in all host certificates.

If the certificate does not have both TLS Web Server Authentication and TLS Web Client Authentication listed in the X509v3 Extended Key Usage section, re-submit the CSR to the CA, and request that they generate a certificate that can be used for both server and client authentication.

3. Inspect the signed certificate to verify that both server and client authentication options are present, as well as the subject alternative name:

```
openssl x509 -in /opt/cloudera/security/pki/$(hostname -f).pem -noout -t
ext
```

Look for output similar to the following to verify the server and client authentication options:

```
 X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentica
tion
```

Look for output similar to the following to validate the subject alternative name:

```
X509v3 Subject Alternative Name:
                DNS:hostname.example.com
```

4. Prepare all the certificates signed by the company's CA on the Cloudera Manager server. In this example, all the certificates are located under the /tmp/auto-tls directory. The password used for keystore and truststore are present in key.pwd and truststore.pwd files respectively.

Important: The password must be more than 12 characters in length and must not include any special characters.

5. Set the ownership of the /tmp/auto-tls directory to cloudera-scm.

```
chown --recursive cloudera-scm:cloudera-scm /tmp/auto-tls
```

6. Refer the example API given below. Customize this API to match the deployment that has been set up and then run the API.

```
curl -i -v -uadmin:admin -X POST --header 'Content-Type: application/json'
 --header 'Accept: application/json' -d '{
"location" : "/opt/cloudera/AutoTLS",
"customCA" : true,
"interpretAsFilenames" : true,
"cmHostCert" : "/tmp/auto-tls/certs/<CM_FQDN>.pem",
"cmHostKey" : "/tmp/auto-tls/keys/<CM_FQDN>-key.pem",
"caCert" : "/tmp/auto-tls/ca-certs/cfssl-chain-truststore.pem",
"keystorePasswd" : "/tmp/auto-tls/keys/key.pwd",
"truststorePasswd" : "/tmp/auto-tls/ca-certs/truststore.pwd",
"trustedCaCerts" : "/tmp/auto-tls/ca-certs.pem",
```

```
"hostCerts" : [ {
"hostname" : "<NODE1_FQDN>",
"certificate" : "/tmp/auto-tls/certs/<NODE1_FQDN>.pem",
"key" : "/tmp/auto-tls/keys/<NODE1_FQDN>-key.pem"
}, {
"hostname" : "<NODE2_FQDN>",
"certificate" : "/tmp/auto-tls/certs/<NODE2_FQDN>.pem",
"key" : "/tmp/auto-tls/keys/<NODE2_FQDN>-key.pem"
}, {
"hostname" : "<NODE3_FQDN>",
"certificate" : "/tmp/auto-tls/certs/<NODE3_FQDN>",
"key" : "/tmp/auto-tls/keys/<NODE3_FQDN>.pem"
} ],
"configureAllServices" : "true",
"sshPort" : 22,
"userName" : "root",
"password" : "cloudera"
}' '<http://<CM_FQDN>>:7180/api/v41/cm/commands/generateCmca'
```

The following section explains each of the API call in detail:

- ```
  curl -i -v -uadmin:admin -X POST --header 'Content-Type: application/jso
  n' --header 'Accept: application/json' -d '
  ```

  Curl command, -d indicates the beginning of content
- ```
  "location" : "/opt/cloudera/AutoTLS",
  ```

  Set a location that cloudera-scm can access and does not already exist (to avoid known issues with picking up old truststores)
- ```
  "customCA" : true,
  "interpretAsFilenames" : true,
  ```

  Setting customCA to false will run AutoTLS use case 1 (CMCA) instead, and does not need to specify any keys, certs or related values
- ```
  "cmHostCert" : "/tmp/auto-tls/certs/ccycloud-7.vcdp71.root.hwx.site.
  pem",
  ```

  The Cloudera Manager host can also be a host in the cluster, so these files will likely need to be referenced again in the hostCerts section. You begin by providing the Certificate and Key for the Cloudera Manager host and other nodes. The certificate can be just the host's signed cert or the entire chain with the host’s signed cert.
- ```
  "cmHostKey" : "/tmp/auto-tls/keys/ccycloud-7.vcdp71.root.hwx.site-key.pe
  m",
  ```

  The key is created initially for hosts along with the CSR. Do not be confused by .key, .crt, .pem suffixes, they are all PEM files.
- ```
  "caCert" : "/tmp/auto-tls/ca-certs/cfssl-chain-truststore.pem",
  ```

  This is the Certificate Authority Certificate Chain, which means that if there are any intermediate CAs, they must be included here and in a certain order. This can be expressed with the command cat inter[n] rootca >

cfssl-chain-truststore.pem. If there are intermediate CA inconsistencies across hosts, include the entire chain with the host's signed certificate for all "certificate" and "cmHostCert" files.

- ```
  "keystorePasswd" : "/tmp/auto-tls/keys/key.pwd",
  ```

  The keystore and truststore do not exist yet, but they must be password protected. CM must be given a password to set for each of them, which is an arbitrary value (a strong password) that is stored in a file. For example, echo strong_password > /tmp/auto-tls/keys/key.pwd

- ```
  \c
  select * from CONFIGS where attr like '%keystore_password';
  select * from CONFIGS where attr like '%truststore_password';
  ```

  The pre-existing passwords may need to be used, which can be found in the backend db, psql.

- ```
  "truststorePasswd" : "/tmp/auto-tls/ca-certs/truststore.pwd",
  ```

  This one can even be set to changeit or changeme, which is default for Java Truststores (see the cacerts file) as this file can be read with or without a password

- trustedCaCerts is completely optional and typically not used. It's a feature that allows you to let CM distribute cacerts across all nodes if they need to act as clients to other organizational CAs. You can safely omit this line:

  "trustedCaCerts" : "/tmp/auto-tls/ca-certs.pem", //This is a path to a PEM file on the Cloudera Manager host which containsa list of CA certificates that should be imported into the truststores of all hosts. This is an optional field.

- This block contains the certificate and key for each host. This is where the naming convention comes in handy to make sure the hostname matches the correct files. All hosts must have certificate and key files provided, including the CM host if it's in the cluster (it usually is)

  ```
  "hostCerts" : [ {
  "hostname" : "ccycloud-7.vcdp71.root.hwx.site",
  "certificate" : "/tmp/auto-tls/certs/ccycloud-7.vcdp71.root.hwx.site.p
  em",
  "key" : "/tmp/auto-tls/keys/ccycloud-7.vcdp71.root.hwx.site-key.pem"
  }, {
  "hostname" : "ccycloud-3.vcdp71.root.hwx.site",
  "certificate" : "/tmp/auto-tls/certs/ccycloud-3.vcdp71.root.hwx.site.
  pem",
  "key" : "/tmp/auto-tls/keys/ccycloud-3.vcdp71.root.hwx.site-key.pem"
  }, {
  "hostname" : "ccycloud-2.vcdp71.root.hwx.site",
  "certificate" : "/tmp/auto-tls/certs/ccycloud-3.vcdp71.root.hwx.site.pem
  ",
  "key" : "/tmp/auto-tls/keys/ccycloud-3.vcdp71.root.hwx.site-key.pem"
  }, {
  "hostname" : "ccycloud-1.vcdp71.root.hwx.site",
  "certificate" : "/tmp/auto-tls/certs/ccycloud-1.vcdp71.root.hwx.site.pe
  m",
  "key" : "/tmp/auto-tls/keys/ccycloud-1.vcdp71.root.hwx.site-key.pem"
  } ],
  ```

- A passwordless sudo user account that exists on all cluster hosts can be used instead of "root" if necessary

  ```
  "configureAllServices" : "true",
  "sshPort" : 22,
  "userName" : "root",
  ```
  ```

- This password should not be cloudera of course, and all hosts must allow NOPASSWD sudo access to remote clients because CM cannot handle a sudo password prompt (it still requires a password for the initial ssh connection).

  ```
  "password" : "cloudera"
  ```

- If a private key is used instead of a password, the "privateKey" option can be used instead of "password", it must be the entire content from the RSA key and must be in the "BEGIN RSA PRIVATE KEY" format. If you cat the key file and it has a different banner, convert to the correct one with:

  ```
  ssh-keygen -p -P "" -N "" -m pem -f <key_path>
  ```

  The key must be on a single line and newlines represented with \n. This command shows the exact output to use as the privateKey value:awk 'NF {sub(/\r/, ""); printf "%s\\n",$0;}' <key_path>
- Change the hostname to point to your CM host (which will need to be https and 7183 if TLS is already enabled)

If a new deployment is set up without TLS encryption, the API uses HTTP and port 7180. If you are converting the deployment to an Auto-TLS setup from an existing Manual TLS setup, the Cloudera Manager UI is converted to HTTPS. In such cases, the URL for the API calls has to be modified.

Note: If you need to use an SSH private key instead of a password, then replace "password" in the above example with "privateKey" and provide the SSH private key as an argument to that field. The SSH private key must be properly JSON-encoded, including replacing newlines with '\n'. The following awk command will output to the terminal the contents of ~/.ssh/id_rsa with newlines replaced, and can be used as input to the "privateKey" argument:

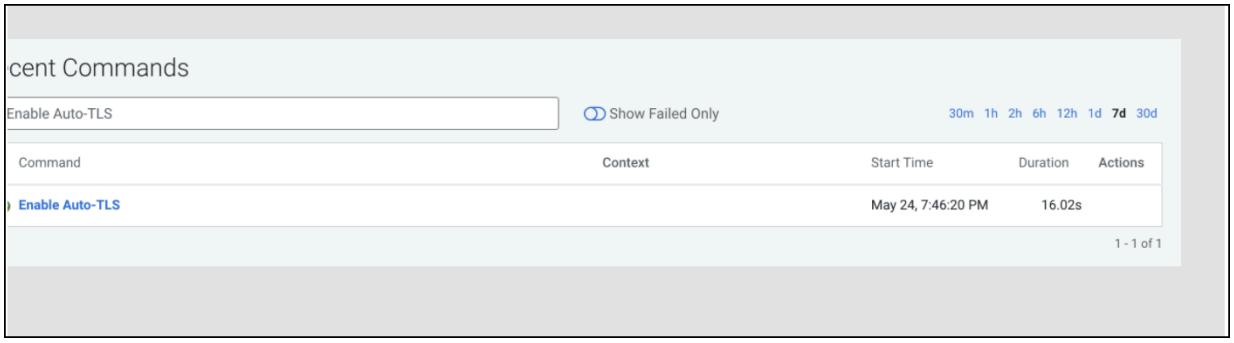
```
awk 'NF {sub(/\r/, ""); printf "%s\\n",$0;}' ~/.ssh/id_rsa
```

Table 2: JSON file key properties

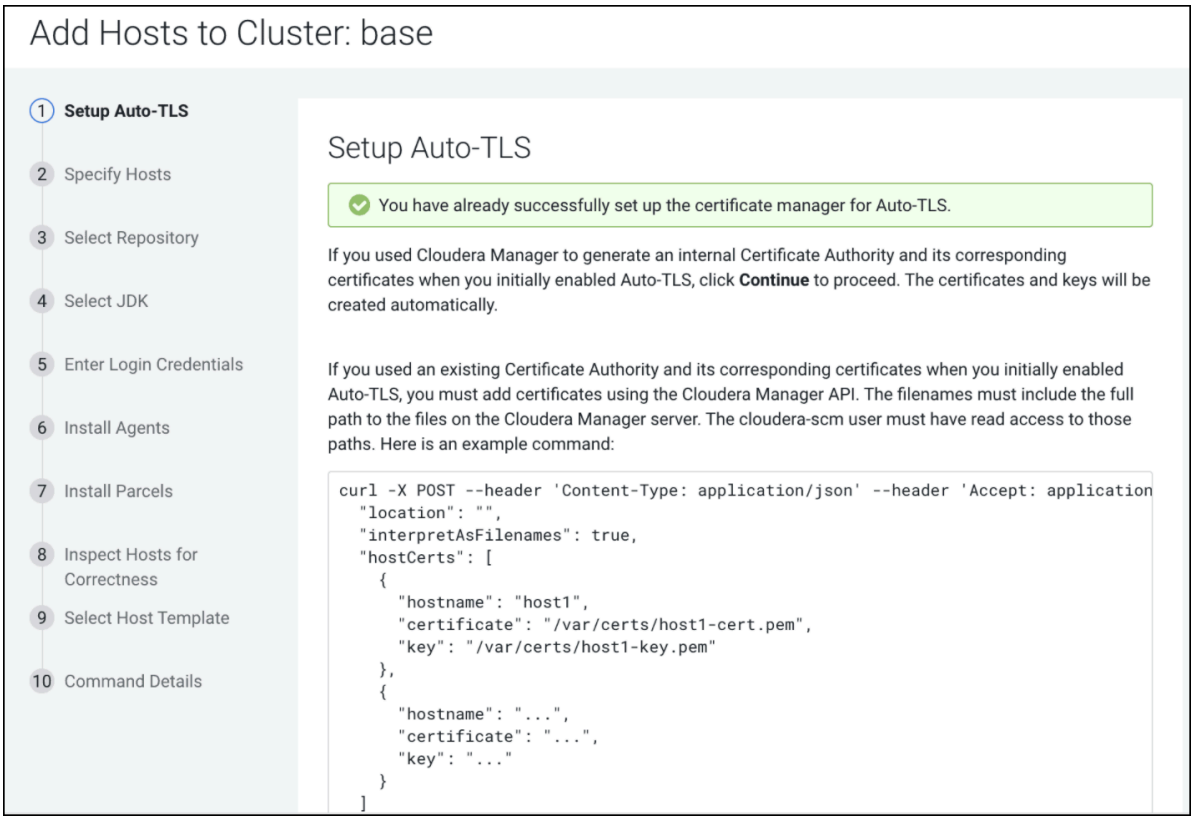
Property	Data type	Description
customCA	boolean	Option to generate an internal Cloudera Manager CA (false) or use user-provided certificates (true). When set to true (user-provided certificates), the following other arguments must be given: * cmHostCert * cmHostKey * caCert * keystorePasswd * truststorePasswd
cmHostCert	string	The certificate for the Cloudera Manager host in PEM format. Only used if customCA == true.
cmHostKey	string	The private key for the Cloudera Manager host in PEM format. Only used if customCA == true.
caCert	string	The certificate for the user-provided certificate authority in PEM format. Only used if customCA == true.
trustedCaCerts	string	A list of CA certificates that will be imported into the Auto-TLS truststore and distributed to all hosts.

Note: If you are using telemetry publisher, then you should ensure that the Starfield CA cert is part of the trusted CA certs bundle.

7. When this API returns successfully, you should see the recent command run as follows.



8. When this API is executing you can check /var/log/cloudera-scm-server/cloudera-scm-server.log for API logs.
9. Restart the Cloudera Manager service. Then restart the Cloudera Manager agents on all cluster servers.
10. The Cloudera Manager UI/API is now available on the TLS port. Now restart all the Cloudera Manager management services.
11. Restart the Cluster services. Now all the services are configured for wire encryption.
12. When adding new hosts to this cluster, the following additional steps need to be performed to upload the CA signed host certificates to Cloudera Manager.
    a. The add hosts wizard will prompt the following screen with instructions to upload the certificates.



    b. Upload the certificates to Cloudera Manager using the following example command:

```
curl -u admin:admin -X POST --header 'Content-Type:
application/json' --header 'Accept: application/json' -d '{
  "location": "/opt/cloudera/AutoTLS",
  "interpretAsFilenames": true,
  "hostCerts": [ {
      "hostname": "ccycloud-10.vcdp71.root.hwx.site",
      "certificate":
"/tmp/auto-tls/certs/ccycloud-10.vcdp71.root.hwx.site.pem",
```

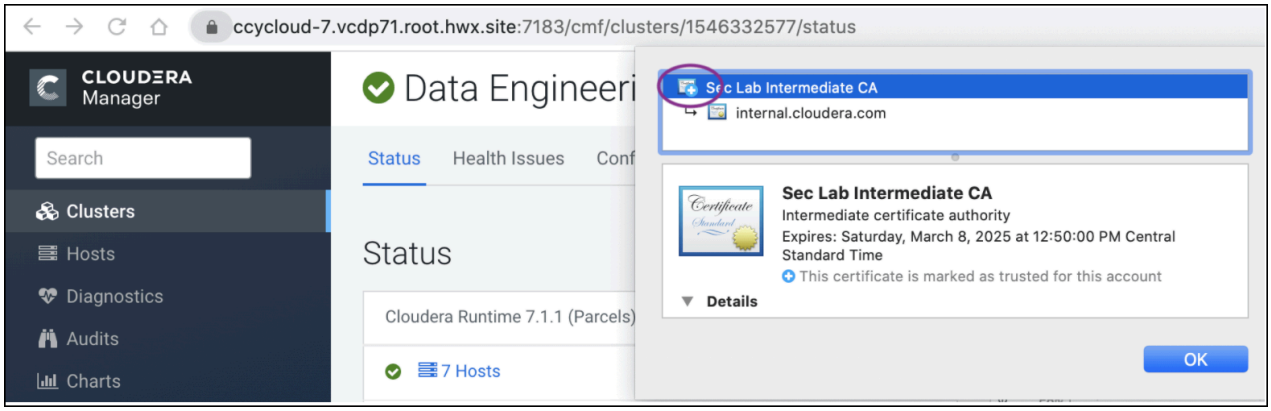
```
      "key":
"/tmp/auto-tls/certs/ccycloud-10.vcdp71.root.hwx.site.pem"
    } ]
}' 'https://ccycloud-7.vcdp71.root.hwx.site:7183/api/v41/cm/commands/ad
dCustomCerts'
```

Note: In the preceding curl command example, the "location" should be of the same value as the location given when the generateCmca API command was run, or omitted if the "location" was not specified when the generateCmca API command was run. Also, the file paths should point to files on the Cloudera Manager server host.

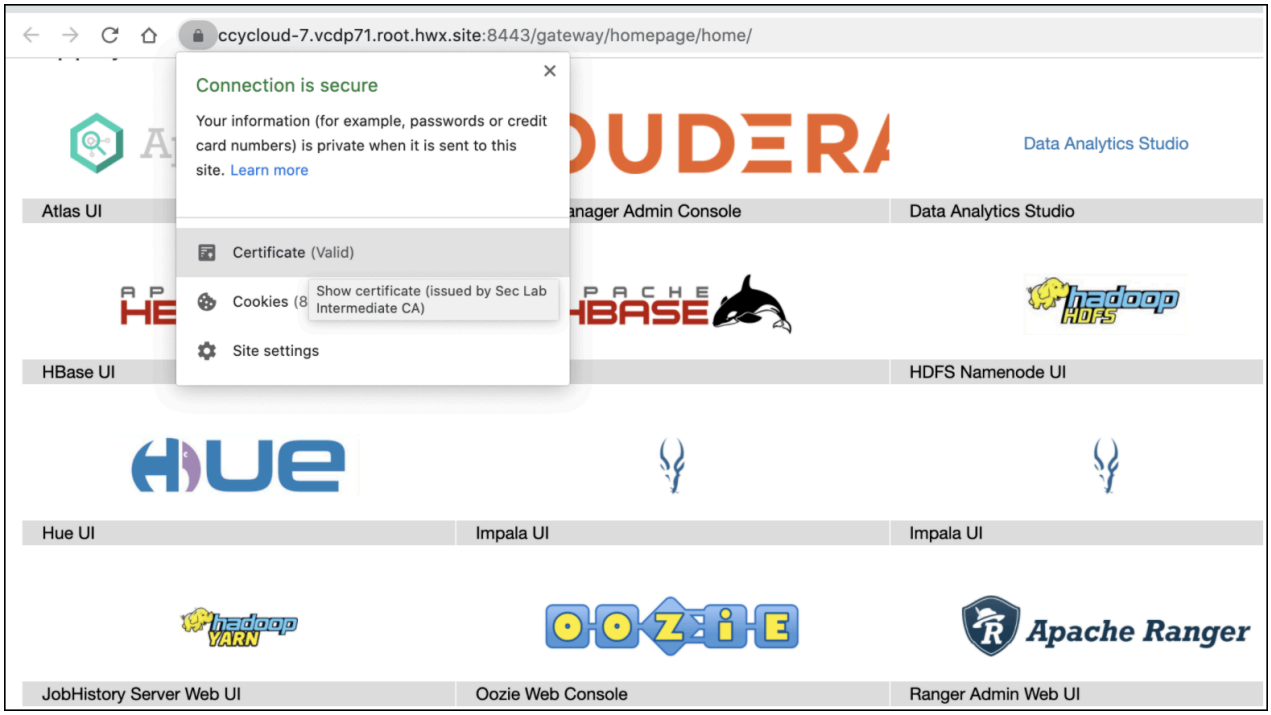
c. Continue to add hosts.

In this example, the CA used to sign all the certificates is Sec Lab Intermediate CA which can be found in the screenshot below:

Cloudera Manager UI:



Knox UI:



Rotate Auto-TLS Certificate Authority and Host Certificates

After the certificate files in the specified paths have been replaced with the new certificates, run the API calls that were used when enabling Auto-TLS. Refer Step 5 in this use case to run the Cloudera Manager API. You do not need to run the `addCustomCerts` API if you are performing the steps given in this use case.

Related Information

Use case 1: Use Cloudera Manager to generate internal CA and corresponding certificates

Use case 2: Enabling Auto-TLS with an intermediate CA signed by an existing Root CA

Manually Configuring TLS Encryption for Cloudera Manager

How to manually enable TLS encryption and certificate authentication for Cloudera Manager.

About this task

When you configure authentication and authorization on a cluster, Cloudera Manager Server sends sensitive information over the network to cluster hosts, such as Kerberos keytabs and configuration files that contain passwords. To secure this transfer, you must configure TLS encryption between Cloudera Manager Server and all cluster hosts.

TLS encryption is also used to secure client connections to the Cloudera Manager Admin Interface, using HTTPS.

Cloudera Manager also supports TLS authentication. Without certificate authentication, a malicious user can add a host to Cloudera Manager by installing the Cloudera Manager Agent software and configuring it to communicate with Cloudera Manager Server. To prevent this, you must install certificates on each agent host and configure Cloudera Manager Server to trust those certificates.

This guide shows how to configure and enable TLS encryption and certificate authentication for Cloudera Manager. The provided examples use an internal certificate authority (CA) to sign all TLS certificates, so this guide also shows you how to establish trust with the CA. (For certificates signed by a trusted public CA, establishing trust is not necessary, because the Java Development Kit (JDK) already trusts them.)

Note: Cloudera recommends using auto-TLS to configure TLS encryption for Cloudera Manager and Cloudera components.

Auto-TLS greatly simplifies the process of enabling and managing TLS encryption on your cluster. It automates the creation of an internal certificate authority (CA) and deployment of certificates across all cluster hosts. It can also automate the distribution of existing certificates, such as those signed by a public CA. Adding new cluster hosts or services to a cluster with auto-TLS enabled creates and deploys the required certificates automatically.

For instructions on enabling auto-TLS, see “Configuring TLS Encryption for Cloudera Manager Using Auto-TLS ”.

Related Information

Configuring TLS Encryption for Cloudera Manager Using Auto-TLS

Generate TLS Certificates

About this task

The following procedure assumes that an internal certificate authority (CA) is used, and shows how to establish trust for that internal CA. If you are using a trusted public CA (such as Symantec, GeoTrust, Comodo, and others), you do

not need to explicitly establish trust for the issued certificates, unless you are using an older JDK and a newer public CA. Older JDKs might not trust newer public CAs by default.

On Each Cluster Host:

About this task

Complete the following steps on each cluster host, including the Cloudera Manager Server host.

Procedure

1. Configure your environment to set JAVA_HOME to the Oracle JDK. For example:

   ```
   export JAVA_HOME=/usr/java/<jdk-version>-cloudera
   ```

   If you log out of the host before completing this procedure, make sure to set JAVA_HOME again when you log in to complete the steps.
2. Create the /opt/cloudera/security/pki directory:

   ```
   sudo mkdir -p /opt/cloudera/security/pki
   ```

   If you choose to use a different directory, make sure you use the same directory on all cluster hosts to simplify management and maintenance.
3. Use the keytool utility to generate a Java keystore and certificate signing request (CSR). Replace the OU, O, L, ST, and C entries with the values for your environment. When prompted, use the same password for the keys tore password and key password. Cloudera Manager does not support using different passwords for the key and keystore.

   ```
   $JAVA_HOME/bin/keytool -genkeypair -alias $(hostname -f) -keyalg
    RSA -keystore /opt/cloudera/security/pki/$(hostname -f).jks -keysiz
   e 2048 -dname "CN=$(hostname -f),OU=ENGINEERING,O=CLOUDERA,L=PALO
    ALTO,ST=CALIFORNIA,C=US" -ext san=dns:$(hostname -f)
   ```

   ```
   $JAVA_HOME/bin/keytool -certreq -alias $(hostname -f) -keystore /opt/clo
   udera/security/pki/$(hostname -f).jks -file /opt/cloudera/security/pki/$
   (hostname -f).csr -ext san=dns:$(hostname -f) -ext EKU=serverAuth,client
   Auth
   ```

   Note: You must ensure that your Issuing Authority will issue the certificates with the extensions Cloudera requires.
4. Submit the CSR files (for example, cm01.example.com.csr) to your certificate authority to obtain a server certificate.

   For security purposes, many commercial CAs ignore requested extensions in a CSR. Make sure that you inform the CA that you require certificates with both server and client authentication options.

   If possible, obtain the certificate in PEM (Base64 ASCII) format. The certificate file is in PEM format if it looks similar to this (some lines omitted):

   ```
   -----BEGIN CERTIFICATE-----
   MIIDAzCCAesCAQAwgY0xCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlh
   MRIwEAYDVQQHEwlQYWxvIEFsdG8xETAPBgNVBAoTCENsb3VkZXJhMRQwEgYDVQQL
   …
   tudY0C32LjGjWOg5ALliN9Oy1u2xRKGAVfapbzAZ2rchtlCZc7mtaT6BXgW8S+Db
   0HhuObn1/8TL4Ho9G+KlJB3MWik2oEbOvQt0rBidMr9qaNX86m0i7pouXZelZ5c5
   UnDPtrhW6A==
   ```

```
-----END CERTIFICATE-----
```

If your issued certificate is in binary (DER) format, convert it to PEM format.

5. After you have received the signed certificate, copy the signed certificate to the following location:

```
/opt/cloudera/security/pki/$(hostname -f).pem
```

6. Inspect the signed certificate to verify that both server and client authentication options are present, as well as the subject alternative name:

```
openssl x509 -in /opt/cloudera/security/pki/$(hostname -f).pem -noout -t
ext
```

Look for output similar to the following to verify the server and client authentication options:

```
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authenticat
ion
```

Look for output similar to the following to validate the subject alternative name:

```
            X509v3 Subject Alternative Name:
                DNS:hostname.example.com
```

Important:

If the certificate does not have the DNS field, re-submit the CSR to the CA, and request that they generate a certificate that keeps the Subject Alternative Name field intact.

If the certificate does not have both TLS Web Server Authentication and TLS Web Client Authentication listed in the X509v3 Extended Key Usage section, re-submit the CSR to the CA, and request that they generate a certificate that can be used for both server and client authentication.

7. Copy the root and intermediate CA certificates to /opt/cloudera/security/pki/rootca.pem and /opt/cloudera/security/pki/intca.pem on each host. If you have a concatenated file containing the root CA and an intermediate CA certificate, split the file along the END CERTIFICATE/BEGIN CERTIFICATE boundary into individual files. If there are multiple intermediate CA certificates, use unique file names such as intca-1.pem, intca-2.pem, and so on.
8. Copy the JDK cacerts file to jssecacerts as follows:

```
sudo cp $JAVA_HOME/jre/lib/security/cacerts $JAVA_HOME/jre/lib/security/
jssecacerts
```

Note: The default password for the cacerts file is changeit. The same applies to the jssecacerts file if you copied it from the cacerts before changing the password. Cloudera recommends changing these passwords by running the following commands:

```
$JAVA_HOME/bin/keytool -storepasswd -keystore $JAVA_HOME/jre/lib/sec
urity/cacerts
```

```
$JAVA_HOME/bin/keytool -storepasswd -keystore $JAVA_HOME/jre/lib/sec
urity/jssecacerts
```

The Oracle JDK uses the jssecacerts file for its default truststore if it exists. Otherwise, it uses the cacerts file. Creating the jssecacerts file allows you to trust an internal CA without modifying the cacerts file that is included with the JDK.

Note: If you upgrade your JDK, make sure to copy your existing jssecacerts file to the new JDK (under $JAVA_HOME/jre/lib/security).

9. Import the root CA certificate into the JDK truststore.

```
sudo $JAVA_HOME/bin/keytool -importcert -alias rootca -keystore $JAVA_HO
ME/jre/lib/security/jssecacerts -file /opt/cloudera/security/pki/rootca.
pem
```

If you see a message like the following, enter yes to continue:

```
Trust this certificate? [no]: yes
```

You must see the following response verifying that the certificate has been properly imported:

```
Certificate was added to keystore
```

10. Perform these steps to replace the self-signed cert with CA issued cert.

```
cd /opt/cloudera/security/pki/; mv $(hostname -f).jks $(hostname -f)-ori
g.jks
```

```
keytool -importkeystore -srcstoretype JKS -deststoretype PKCS12 -srckeys
tore $(hostname -f)-orig.jks -srcalias $(hostname -f) -destkeystore $(ho
stname -f).p12
```

```
openssl pkcs12 -in $(hostname -f).p12 -nodes -nocerts -out $(hostname -f)-
pk.pem
```

```
openssl pkcs12 -export -in $(hostname -f).pem -inkey $(hostname -f)-pk.pem
 -name $(hostname -f) -out $(hostname -f).pk12
```

```
keytool -importkeystore -deststoretype JKS -srcstoretype PKCS12 -srckeys
tore  $(hostname -f).pk12 -destkeystore $(hostname -f).jks
```

11. Append the intermediate CA certificate to the signed host certificate, and then import it into the keystore. Make sure that you use the append operator (>>) and not the overwrite operator (>):

```
sudo cat /opt/cloudera/security/pki/intca.pem >> /opt/cloudera/security/
pki/$(hostname -f).pem
```

12. Create symbolic links (symlink) for the certificate and keystore files:

```
sudo ln -s /opt/cloudera/security/pki/$(hostname -f).pem /opt/cloudera/s
ecurity/pki/agent.pem
```

This allows you to use the same /etc/cloudera-scm-agent/config.ini file on all agent hosts rather than maintaining a file for each agent.

On the Cloudera Manager Server Host

About this task

On the Cloudera Manager Server host, create an additional symlink for the keystore file:

```
sudo ln -s /opt/cloudera/security/pki/$(hostname -f).jks /opt/cloudera/secur
ity/pki/server.jks
```

Configure TLS for the Cloudera Manager Admin Console

About this task

Minimum Required Role: Cluster Administrator (also provided by Full Administrator)

Use the following procedure to enable TLS encryption for the Cloudera Manager Server admin interface. Make sure you have generated the host certificate as described in “Manually Configuring TLS Encryption for Cloudera Manager”>“On Each Cluster Host”.

Step 1: Enable HTTPS for the Cloudera Manager Admin Console

Procedure

1. Log in to the Cloudera Manager Admin Console.
2. Select Administration Settings .
3. Select the Security category.
4. Configure the following TLS settings:

Property	Description
Cloudera Manager TLS/SSL Server JKS Keystore File Location	The complete path to the keystore file. For example: `/opt/cloudera/security/pki/server.jks`
Cloudera Manager TLS/SSL Server JKS Keystore File Password	The password for the /opt/cloudera/security/jks/server.jks keystore.
Use TLS Encryption for Admin Console	Check this box to enable TLS encryption for Cloudera Manager.

5. Enter a Reason for Change, then click Save Changes to save the settings.

Step 2: Specify SSL Truststore Properties for Cloudera Management Services

About this task

When enabling TLS for the Cloudera Manager Server admin interface, you must set the Java truststore location and password in the Cloudera Management Services configuration. Otherwise, roles such as Host Monitor and Service Monitor cannot connect to Cloudera Manager Server and will not start.

Configure the path and password for the $JAVA_HOME/jre/lib/security/jssecacerts truststore that you created earlier. Make sure that you have created this file on all hosts, including the Cloudera Management Service hosts, as instructed in “Manually Configuring TLS Encryption for Cloudera Manager”>“On Each Cluster Host”.

Procedure

1. Open the Cloudera Manager Administration Console and go to the Cloudera Management Service service.
2. Click the Configuration tab.
3. Select Scope Cloudera Management Service (Service-Wide) .
4. Select Category Security .

5. Edit the following TLS/SSL properties according to your cluster configuration.

Property	Description
TLS/SSL Client Truststore File Location	The path to the client truststore file used in HTTPS communication. This truststore contains certificates of trusted servers, or of Certificate Authorities trusted to identify servers. For this example, set the value to: `<JAVA_HOME>/jre/lib/security/jssecacerts` Replace <*JAVA_HOME*> with the path to the Oracle JDK.
Cloudera Manager Server TLS/SSL Certificate Trust Store Password	The password for the truststore file.

6. Enter a Reason for Change, then click Save Changes to save the settings.

Step 3: Restart Cloudera Manager and Services

About this task

You must restart both Cloudera Manager Server and the Cloudera Management Service for TLS encryption to work. Otherwise, the Cloudera Management Services (such as Host Monitor and Service Monitor) cannot communicate with Cloudera Manager Server.

Procedure

1. Restart the Cloudera Manager Server by running the following command on the Cloudera Manager Server host:
   - RHEL 7 compatible:

     ```
     sudo systemctl restart cloudera-scm-server
     ```
   - RHEL 6 compatible, SLES, Ubuntu:
   -

     ```
     sudo service cloudera-scm-server restart
     ```
2. After the restart completes, connect to the Cloudera Manager Admin Console using the HTTPS URL (for example: https://*CM01.EXAMPLE.COM*:7183). If you used an internal CA-signed certificate, you must configure your browser to trust the certificate. Otherwise, you will see a warning in your browser any time you access the Cloudera Manager Administration Console. By default, certificates issued by public commercial CAs are trusted by most browsers, and no additional configuration is necessary if your certificate is signed by one of them.
3. Restart the Cloudera Management Service (Cloudera Management Service Actions Restart).

Configure TLS for Cloudera Manager Agents

About this task

Minimum Required Role: Cluster Administrator (also provided by Full Administrator)

Use the following procedure to encrypt the communication between Cloudera Manager Server and Cloudera Manager Agents:

Step 1: Enable TLS Encryption for Agents in Cloudera Manager

About this task

Configure the TLS properties for Cloudera Manager Agents.

Procedure

1. Log in to the Cloudera Manager Admin Console.
2. Select Administration Settings .
3. Select the Security category.
4. Select the Use TLS Encryption for Agents option.
5. Enter a Reason for Change, then click Save Changes to save the settings.

Step 2: Enable TLS on Cloudera Manager Agent Hosts

About this task

To enable TLS between the Cloudera Manager agents and Cloudera Manager, you must specify values for the TLS properties in the /etc/cloudera-scm-agent/config.ini configuration file on all agent hosts.

Procedure

- On each agent host (including the Cloudera Manager Server host, which also has an agent), open the /etc/cloudera-scm-agent/config.ini configuration file and set the use_tls parameter in the [Security] section as follows:

  ```
  use_tls=1
  ```

  Alternatively, you can edit the config.ini file on one host, and then copy it to the other hosts because this file by default does not contain host-specific information. If you have modified properties such as listening_hostname or listening_ip address in config.ini, you must edit the file individually on each host.

Step 3: Restart Cloudera Manager Server and Agents

Procedure

1. Restart the Cloudera Manager Server by running the following command on the Cloudera Manager Server host:
   - RHEL 7 compatible:

     ```
     sudo systemctl restart cloudera-scm-server
     ```
   - RHEL 6 compatible, SLES, Ubuntu:
   -
     ```
     sudo service cloudera-scm-server restart
     ```
2. On each agent host (including the Cloudera Manager Server host), restart the Cloudera Manager agent service:
   - RHEL 7 compatible:

     ```
     sudo systemctl restart cloudera-scm-agent
     ```
   - RHEL 6 compatible, SLES, Ubuntu:
   -
     ```
     sudo service cloudera-scm-agent restart
     ```

Step 4: Verify that the Cloudera Manager Server and Agents are Communicating

About this task

In the Cloudera Manager Admin Console, go to Hosts All Hosts . If you see successful heartbeats reported in the Last Heartbeat column after restarting the agents, TLS encryption is working properly.

Enable Server Certificate Verification on Cloudera Manager Agents

About this task

Minimum Required Role: Cluster Administrator (also provided by Full Administrator)

If you have completed the previous sections, communication between Cloudera Manager server and the agents is encrypted, but the certificate authenticity is not verified. For full security, you must configure the agents to verify the Cloudera Manager server certificate. If you are using a server certificate signed by an internal certificate authority (CA), you must configure the agents to trust that CA:

Procedure

1. On each agent host (including the Cloudera Manager Server host), open the /etc/cloudera-scm-agent/config.ini configuration file, and then uncomment and set the following property:

   ```
   verify_cert_file=/opt/cloudera/security/pki/rootca.pem
   ```

   Alternatively, you can edit the config.ini file on one host, and then copy it to the other hosts because this file by default does not contain host-specific information. If you have modified properties such as listening_hostname or listening_ip address in config.ini, you must edit the file individually on each host.
2. Restart the Cloudera Manager agents. On each agent host (including the Cloudera Manager Server host), run the following command:
   - RHEL 7 compatible:

     ```
     sudo systemctl restart cloudera-scm-agent
     ```
   - RHEL 6 compatible, SLES, Ubuntu:
   -
     ```
     sudo service cloudera-scm-agent restart
     ```
3. Verify that the Cloudera Manager server and agents are communicating. In the Cloudera Manager Admin Console, go to Hosts All Hosts . If you see successful heartbeats reported in the Last Heartbeat column after restarting the agents and management service, TLS verification is working properly. If not, check the agent log (/var/log/cloudera-scm-agent/cloudera-scm-agent.log) for errors.

Configure Agent Certificate Authentication

About this task

Important: Perform this procedure on each agent host, including the Cloudera Manager Server host, which also has an agent.

Without certificate authentication, a malicious user can add a host to Cloudera Manager by installing the Cloudera Manager agent software and configuring it to communicate with Cloudera Manager Server. To prevent this, you must configure Cloudera Manager to trust the agent certificates.

Step 1: Export the Private Key to a File

About this task

On each Cloudera Manager Agent host, use the keytool utility to export the private key and certificate to a PKCS12 file, which can then be split up into individual key and certificate files using the openssl command:

Procedure

1. Export the private key and certificate:

```
sudo $JAVA_HOME/bin/keytool -importkeystore -srckeystore /opt/cloudera/s
ecurity/pki/$(hostname -f).jks -destkeystore /opt/cloudera/security/pki/
$(hostname -f)-key.p12 -deststoretype PKCS12 -srcalias $(hostname -f)
```

2. Use the openssl command to export the private key into its own file:

```
sudo openssl pkcs12 -in /opt/cloudera/security/pki/$(hostname -f)-key.p12
 -nocerts -out /opt/cloudera/security/pki/$(hostname -f).key
```

3. Create a symbolic link for the .key file:

```
sudo ln -s /opt/cloudera/security/pki/$(hostname -f).key /opt/cloudera/s
ecurity/pki/agent.key
```

This allows you to use the same /etc/cloudera-scm-agent/config.ini file on all agent hosts rather than maintaining a file for each agent.

Step 2: Create a Password File

About this task

The Cloudera Manager agent obtains the password from a text file, not from a command line parameter or environment variable. The password file allows you to use file permissions to protect the password. For example, run the following commands on each Cloudera Manager Agent host, or run them on one host and copy the file to the other hosts:

Create and secure the file containing the password used to protect the private key of the Agent:

Procedure

1. Use a text editor to create a file called /etc/cloudera-scm-agent/agentkey.pw that contains the password.
2. Change ownership of the file to root:

```
sudo chown root:root /etc/cloudera-scm-agent/agentkey.pw
```

3. Change the permissions of the file:

```
sudo chmod 440 /etc/cloudera-scm-agent/agentkey.pw
```

Step 3: Configure the Agent to Use Private Keys and Certificates

About this task

On a Cloudera Manager Agent, open the /etc/cloudera-scm-agent/config.ini configuration file, uncomment and edit the following properties:

Property	Example Value	Description
client_key_file	/opt/cloudera/security/pki/agent.key	Path to the private key file.
client_keypw_file	/etc/cloudera-scm-agent/agentkey.pw	Path to the private key password file.
client_cert_file	/opt/cloudera/security/pki/agent.pem	Path to the client certificate file.

Copy the file to all other cluster hosts. If you have modified properties such as listening_hostname or listening_ip address in config.ini, you must edit the file individually on each host.

Step 4: Enable Agent Certificate Authentication

Procedure

1. Log in to the Cloudera Manager Admin Console.
2. Select Administration Settings .
3. Click the Security category.
4. Configure the following TLS settings:

Setting	Description
Use TLS Authentication of Agents to Server	Select this option to enable TLS authentication of agents to the server.
Cloudera Manager TLS/SSL Certificate Trust Store File	Specify the full filesystem path to the jssecacerts file located on the Cloudera Manager Server host. For this example, set the value to: `<JAVA_HOME>/jre/lib/security/jssecacerts` Replace <*JAVA_HOME*> with the path to the Oracle JDK.
Cloudera Manager TLS/SSL Certificate Trust Store Password	Specify the password for the jssecacerts truststore.

5. Enter a Reason for Change, then click Save Changes to save the settings.

Step 5: Restart Cloudera Manager Server and Agents

Procedure

1. On the Cloudera Manager server host, restart the Cloudera Manager server:
   - RHEL 7 compatible:

     ```
     sudo systemctl restart cloudera-scm-server
     ```
   - RHEL 6 compatible, SLES, Ubuntu:
   -
     ```
     sudo service cloudera-scm-server restart
     ```
2. On every agent host, restart the Cloudera Manager agent:
   - RHEL 7 compatible:

     ```
     sudo systemctl restart cloudera-scm-agent
     ```
   - RHEL 6 compatible, SLES, Ubuntu:
   -
     ```
     sudo service cloudera-scm-agent restart
     ```

Step 6: Verify that Cloudera Manager Server and Agents are Communicating

About this task

In the Cloudera Manager Admin Console, go to Hosts All Hosts . If you see successful heartbeats reported in the Last Heartbeat column after restarting the agents and server, TLS certificate authentication is working properly. If not, check the agent log (/var/log/cloudera-scm-agent/cloudera-scm-agent.log) for errors.

For example, you might see the following error:

```
WrongHost: Peer certificate commonName does not match host, expected 192.0.2
.155, got cdh-1.example.com
[02/May/2018 15:04:15 +0000] 4655 MainThread agent       ERROR    Heartbeat
ing to 192.0.2.155:7182 failed
```

For this scenario, make sure that your DNS and /etc/hosts file are configured correctly, and that your server_host parameter in /etc/cloudera-scm-agent/config.ini uses the Cloudera Manager Server hostname, and not IP address.

Disable weak ciphers for TLS servers

You can disable weak ciphers for TLS servers.

To disable ciphers, append :!<cipher_name> to both cipher_list and server_cipher_list in the /etc/cloudera-scm-agent/config.ini file.

The default values of those cipher configurations in the config.ini file are:

```
cipher_list=HIGH:!DSS:!DH:!ADH:!DES:!3DES:!SHA1:!aNULL:!eNULL:!EXPORT:!SSLv2
:!SSLv3:!TLSv1
```

```
server_cipher_list=HIGH:!DSS:!DH:!ADH:!DES:!3DES:!SHA1:!aNULL:!eNULL:!EXPORT
:!SSLv2:!SSLv3:!TLSv1:!CAMELLIA
```

Append the following to each of the values for cipher_list and server_cipher_list:

```
:!ECDHE-RSA-AES256-SHA384:!ECDHE-RSA-AES128-SHA256:!AES256-CCM8:!AES256-CCM:
!AES128-CCM8:!AES128-CCM:!AES256-SHA256:!AES128-SHA256
```

Configuring TLS/SSL encryption manually for Cloudera Services

To configure TLS/SSL encryption manually for Cloudera services.

Configuring TLS encryption manually for Apache Atlas

Configuring Apache Atlas Transport Layer Security (TLS) involves both one-way (server authentication) and two-way (server and client authentication).

Procedure

1. Create a keystore file:

```
cd /usr/java/<jdk-version>/bin/
keytool -genkey -alias serverkey -keypass <keypass> -keyalg RSA
-sigalg SHA1withRSA -keystore atlas.keystore -storepass <keypass>
-validity 3650 -dname "CN=Nicola Marangoni, OU=PS, O=Hortonworks,
L=Munich, ST=BY, C=DE"
```

2. Select Atlas > Configs > Advanced, and select Advanced application-properties and set the following properties:

Property	Value	Description
atlas.enableTLS	true	Enable or disable TLS listener. Set this value to true to enable TLS (default value is false).

Add the following properties by selecting Custom application-properties > Add Property.

Property	Value	Description
keystore.file	/home/atlas/atlas.keystore	The path to the keystore file leveraged by the server. This file contains the server certificate.
truststore.file	/home/atlas/atlas.keystore	The path to the truststore file. This file contains the certificates of other trusted entities (for example, the certificates for client processes if two-way TLS is enabled). In most instances this can be set to the same value as the keystore.file property (especially if one-way TLS is enabled).
atlas.ssl.exclude.cipher .suites	.*NULL.*, .*RC4.*, .*MD5.*, .*DES.*, .*DSS.*	The excluded Cipher Suites list - .*NULL.*,.*RC4.*,.*MD5.*,.*DES.*,.*DSS.* are weak and unsafe Cipher Suites that are excluded by default. If additional Ciphers need to be excluded, set this property with the default Cipher Suites such as .*NULL.*, .*RC4.*, .*MD5.*, .*DES.*, .*DSS.*, and add the additional Cipher Suites to the list with a comma separator. They can be added with their full name or a regular expression. The Cipher Suites listed in the atlas.ssl.exclude.cipher.suites property take precedence over the default Cipher Suites. You should retain the default Cipher Suites, and add additional ones to increase security.

Note: If the user wants to add any Cipher Suites, they must manually add it through the Atlas Server Advanced Configuration Snippet (Safety Valve) for conf/atlas-application.properties.

The default HTTP port is 31000 and the default HTTPS port is 31443. These values can be overridden using the atlas.server.http.port and atlas.server.https.port properties, respectively.

3. Select Actions > Restart on the Cloudera Manager instance to restart all services.

   If you disable Atlas TLS, you must clear your browser cookies to log in to the Atlas UI using HTTP request headers.

Enable security for Cruise Control

When AutoTLS is disabled, you need to configure the security properties in Cloudera Manager to use Cruise Control in a secure environment. You can also choose between SPENGO and Trusted Proxy as an authentication method, and can assign admin, user and viewer roles to users to achieve further authorization over Cruise Control tasks.

About this task

You can use TLS/SSL security protocols for securing Cruise Control. You must set the TLS/SSL security protocol in Cruise Control just as it is set for Kafka. When TLS/SSL security is enabled, the secure connection is automatically set for Zookeeper as well for secure communication and the non-secure ports are cannot be used.

There are two authentication methods for Cruise Control: SPENGO and Trusted Proxy. SPENGO uses Kerberos over HTTP. Trusted Proxy uses Knox through a gateway mechanism where Knox authenticates with Cruise Control over SPENGO and forwards the real user ID.

Before you begin

Ensure that you have set up TLS for Cloudera Manager:

- Generate TLS certificates
- Configure TLS for Admin Console and Agents
- Enable server certificate verification on Agents
- Configure agent certificate authentication

Procedure

1. Access for the Cruise Control configurations.
   a) Go to your cluster in .
   b) Select Cruise Control from the list of Services.
   c) Click on Configuration tab.
2. Select Category > Main.
3. Edit the authorization and Kafka security properties according to the cluster configuration.
   You need the following authorization and security properties from the Main category.

Security property		Description
Kafka Client Security Protocol	security.protocol	Protocol to be used for communicating with Kafka. Select the same protocol as you use for Kafka.
Authentication Method	auth_method	Authentication method that Cruise Control uses to authenticate clients.
Trusted Proxy Authentication Service	trusted_auth_service_user	The username part of the trusted proxy authentication service's principal. The default service is Knox for Cruise Control.
ADMIN Level Users	auth_admins	The list of ADMIN level users to have access to all endpoints.
USER Level Users	auth_users	The list of USER level users to have access to all the GET endpoints except bootstrap and train.
VIEWER Level Users	auth_viewers	The list of VIEWER level users to have access to the most lightweight kafka_cl uster_state, user_tasks and review_board endpoints.

4. Click Save Changes.
5. Click Clear next to the Category Filter.
6. Select Category > Security.
   All the security related properties are displayed.
7. Edit the security properties according to the cluster configuration.

Security property		Description
Enable TLS/SSL for Cruise Control Server	webserver.ssl.enable	Encrypting communication between clients and Cruise Control Server using TLS.
Cruise Control Server TLS/SSL Server Keystore File Location	webserver.ssl.keystore.location	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Cruise Control Server is acting as a TLS/SSL server.
Cruise Control Server TLS/SSL Server Keystore File Password	webserver.ssl.keystore.password	The password for the Cruise Control Server keystore file.
Cruise Control Server TLS/SSL Server Keystore Key Password	webserver.ssl.key.password	The password that protects the private key contained in the keystore used when Cruise Control Server is acting as a TLS/SSL server.

Security property		Description
HTTP Strict Transport Security Enabled	webserver.ssl.sts.enabled	Enables the Strict Transport Security header in the web server responses. By default the configuration is enabled.
Cruise Control Server TLS/SSL Client Trust Store File	ssl.truststore.location	The location on disk of the trust store, used to confirm the authenticity of TLS/SSL servers that Cruise Control Server might connect to. This is used when Cruise Control Server is the client in a TLS/SSL connection. 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.
Cruise Control Server TLS/SSL Client Trust Store Password	ssl.truststore.password	The password for the Cruise Control Server TLS/SSL Certificate Trust Store File. This password is not required to access the trust store, the 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.

8. Click Save Changes.
9. Click Clear next to the Category Filter.
10. Type ssl.properties to the Search field.

    The Cruise Control Server Advanced Configuration Snippet (Safety Valve) for ssl.properties field is displayed. You can define additional confiuration parameters for Cruise Control in the Safety Valve field. You also need to add properties that are inherited from Kafka clients, if needed for the secured communication. The values for the inherited properties should match with the values defined in Kafka.

    a) Add the webserver keystore type, if needed, by adding the following property and its value to the Advanced Configuration Snippet field.

Security property	Description
webserver.ssl.keystore.type	The file format of the key store file. This configuration is optional. If the keystore type for the webserver is not set, it falls back to Jetty's default behavior.

    b) Add the following additional security properties to the Advanced Configuration Snippet.

Security property	Description
ssl.provider	The name of the security provider used for SSL connections. Default value 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 TLS or SSL network protocol.
ssl.enabled.protocols	This property should list at least one of the protocols (TLSv1.2, TLSv1.1,TLSv1) configured on the broker side.
ssl.truststore.type	The file format of the trust store file.
ssl.keystore.type	The file format of the key store file.

11. Adding the following properties to the Advanced Configuration Snippet helps customizing the default TLS configurations defined for the Cruise Control web server:

Security property	Description
webserver.ssl.include.ciphers	Sets the included ciphers for the webserver.
webserver.ssl.exclude.ciphers	Sets the excluded ciphers for the webserver.

Security property	Description
webserver.ssl.include.protocols	Sets the included protocols for the webserver.
webserver.ssl.exclude.protocols	Sets the excluded protocols for the webserver.

12. Click Save Changes.

Configuring TLS/SSL for HBase

Once all the prerequisites are fulfilled, you can configure TLS/SSL for HBase Web UIs, HBase REST Server and HBase Thrift Server.

Prerequisites to configure TLS/SSL for HBase

Before configuring TLS/SSL for HBase, ensure that all prerequisites are fulfilled.

- Before enabling TLS/SSL, ensure that keystores containing certificates bound to the appropriate domain names will need to be accessible on all hosts on which at least one HBase daemon role is running.
- Keystores for HBase must be owned by the hbase group, and have permissions 0440 (that is, readable by owner and group).
- You must specify absolute paths to the keystore and truststore files. These settings apply to all hosts on which daemon roles of the HBase service run. Therefore, the paths you choose must be valid on all hosts.
- Cloudera Manager supports the TLS/SSL configuration for HBase at the service level. Ensure you specify absolute paths to the keystore and truststore files. These settings apply to all hosts on which daemon roles of the service in question run. Therefore, the paths you choose must be valid on all hosts.

  An implication of this is that the keystore file names for a given service must be the same on all hosts. If, for example, you have obtained separate certificates for HBase daemons on hosts node1.example.com and node2.example.com, you might have chosen to store these certificates in files called hbase-node1.keystore and hbase-node2.keystore (respectively). When deploying these keystores, you must give them both the same name on the target host — for example, hbase.keystore.

Configuring TLS/SSL for HBase Web UIs

You can configure TLS/SSL for HBase Web UIs using Cloudera Manager.

Procedure

1. In Cloudera Manager, select the HBase service.
2. Click the Configuration tab.
3. Use the Scope / HBase (Service-Wide) filter.
4. Search for tls/ssl.
5. Check Web UI TLS/SSL Encryption Enabled.
6. Edit the HBase TLS/SSL properties according to your configuration.

Table 3: HBase TLS/SSL Properties

Property	Description
HBase TLS/SSL Server JKS Keystore File Location	Path to the keystore file containing the server certificate and private key used for encrypted web UIs.
HBase TLS/SSL Server JKS Keystore File Password	Password for the server keystore file used for encrypted web UIs.
HBase TLS/SSL Server JKS Keystore Key Password	Password that protects the private key contained in the server keystore used for encrypted web UIs.

7. Click Save Changes.
8. Restart the HBase service.

Configuring TLS/SSL for HBase REST Server

You can configure TLS/SSL for HBase REST Server using Cloudera Manager.

Procedure

1. In Cloudera Manager, select the HBase service.
2. Click the Configuration tab.
3. Search for tls/ssl rest.
4. Check Enable TLS/SSL for HBase REST Server.
5. Edit the HBase REST Serer TLS/SSL properties according to your configuration.

Table 4: HBase TLS/SSL Properties

Property	Description
HBase REST Server 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 HBase REST Server is acting as a TLS/SSL server. The keystore must be in JKS format.file.
HBase REST Server TLS/SSL Server JKS Keystore File Password	The password for the HBase REST Server JKS keystore file.
HBase REST Server TLS/SSL Server JKS Keystore Key Password	The password that protects the private key contained in the JKS keystore used when HBase REST Server is acting as a TLS/SSL server.

6. Click Save Changes.
7. Restart the HBase service.

Configuring TLS/SSL for HBase Thrift Server

You can configure TLS/SSL for HBase Thrift Server using Cloudera Manager.

Procedure

1. In Cloudera Manager, select the HBase service.
2. Click the Configuration tab.
3. Search for tls/ssl thrift.
4. Check Enable TLS/SSL for HBase Thrift Server over HTTP.
5. Edit the HBase REST Server TLS/SSL properties according to your configuration.

Table 5: HBase TLS/SSL Properties

Property	Description
HBase Thrift Server over HTTP TLS/SSL Server JKS Keystore File Location	Path to the TLS/SSL keystore file (in JKS format) with the TLS/SSL server certificate and private key. Used when HBase Thrift Server over HTTP acts as a TLS/SSL server.
HBase Thrift Server over HTTP TLS/SSL Server JKS Keystore File Password	Password for the HBase Thrift Server JKS keystore file.
HBase Thrift Server over HTTP TLS/SSL Server JKS Keystore Key Password	Password that protects the private key contained in the JKS keystore used when HBase Thrift Server over HTTP acts as a TLS/SSL server.

6. Click Save Changes.
7. Restart the HBase service.

Enabling TLS/SSL for HiveServer

You can secure client-server communications using symmetric-key encryption in the TLS/SSL (Transport Layer Security/Secure Sockets Layer) protocol. To encrypt data exchanged between HiveServer and its clients, you can use Cloudera Manager to configure TLS/SSL.

Before you begin

- HiveServer has the necessary server key, certificate, keystore, and trust store set up on the host system.
- The hostname variable ($(hostname -f)-server.jks) is used with Java keytool commands to create keystore, as shown in this example:

```
$ sudo keytool -genkeypair -alias $(hostname -f)-server -keyalg RSA -key
store \
    /opt/cloudera/security/pki/$(hostname -f)-server.jks -keysize 2048 -
dname \
    "CN=$(hostname -f),OU=DEPT-NAME-OPTIONAL,O=COMPANY-
NAME,L=CITY,ST=STATE,C=TWO-DIGIT-NATION" \
    -storepass PASSWORD -keypass PASSWORD
```

About this task

On the beeline command line, the JDBC URL requirements include specifying ssl=true;sslTrustStore=<path_to_truststore>. Truststore password requirements depend on the version of Java running in the cluster:

- Java 11: the truststore format has changed to PKCS and the truststore password is required; otherwise, the connection fails.
- Java 8: The trust store password does not need to be specified.

Procedure

1. In Cloudera Manager, navigate to Clusters Hive Configuration .
2. In Filters, select HIVE for the scope.
3. Select Security for the category.
4. Accept the default Enable TLS/SSL for HiveServer2, which is checked for Hive (Service-Wide).
5. Enter the path to the Java keystore on the host system.
   /opt/cloudera/security/pki/*KEYSTORE_NAME*.jks
6. Enter the password for the keystore you used on the Java keytool command-line when the key and keystore were created.

   The password for the keystore must match the password for the key.
7. Enter the path to the Java trust store on the host system.
8. Click Save Changes.
9. Restart the Hive service.
10. Construct a connection string for encrypting communications using TLS/SSL.

```
jdbc:hive2://#<host>:#<port>/#<dbName>;ssl=true;sslTrustStore=#<ssl_trus
tstore_path>; \
trustStorePassword=#<truststore_password>;#<otherSessionConfs>?#<hiveCon
fs>#<hiveVars>
```

Configuring TLS/SSL for Cloudera Data Explorer (Hue)

You can independently enable TLS/SSL for Data Explorer.

Cloudera recommends that your cluster and the Data Explorer service use Kerberos for authentication. If you enable TLS/SSL for a cluster that has not been configured to use Kerberos, a warning is displayed. You should integrate the cluster with your Kerberos deployment before proceeding.

Creating a truststore file in PEM format

You must create the Data Explorer Truststore by consolidating certificates of all SSL-enabled servers (or a single CA Certificate chain) that Data Explorer communicates with into one file. This generally includes certificates of all the Oozie, HDFS, MapReduce, and YARN daemons, and any other SSL-enabled services.

About this task

Server certificates are stored in Java KeyStore (JKS) format. The Data Explorer Truststore must be in the Privacy Enhanced Mail (PEM) format whereas other services use the JKS format by default. To create the Data Explorer truststore, extract each certificate from Hadoop's Java Keystore with the Java keytool, convert the certificate to PEM format with the OpenSSL.org openssl tool, and then add it to the Data Explorer truststore:

Procedure

1. Extract the certificate from the keystore of each TLS/SSL-enabled server with which Data Explorer communicates. For example, if you have hadoop-server.keystore that contains a server certificate, foo-1.example.com with a password of example123, you would use the following keytool command:

   ```
   keytool -exportcert -keystore hadoop-server.keystore -alias foo-1.example.com -storepass example123 -file foo-1.cert
   ```

2. Convert each certificate into a PEM file. Here is what the openssl tool command looks like for the foo-1.cert file that was extracted in Step 1:

   ```
   openssl x509 -inform der -in foo-1.cert > foo-1.pem
   ```

3. Concatenate all the PEM certificates you extracted and converted from the server truststore into one PEM file:

   ```
   cat foo-1.pem foo-2.pem foo-N.pem ... > hue_truststore.pem
   ```

   Concatenate the certificate files in the following order: SSL certificate followed by intermediate certificate, followed by the root CA certificate.

   Important: Ensure the final PEM truststore is deployed in a location that is accessible by the Data Explorer service.

4. Log in to Cloudera Manager as an Administrator.
5. Go to Clusters Hue Configuration and add the following line in the Hue Service Environment Advanced Configuration Snippet (Safety Valve) field:

   ```
   REQUESTS_CA_BUNDLE=[***PATH-TO-HUETRUST.PEM-FIL E***]
   ```

6. Click Save Changes.
7. Restart the Data Explorer service.

Configuring Cloudera Data Explorer (Hue) as a TLS/SSL client

Data Explorer acts as a TLS/SSL client when communicating with other services, such as core Hadoop, HBase, Oozie, and cloud providers like Amazon S3 or Azure.

To act as a TLS/SSL client, Data Explorer must authenticate HDFS, MapReduce, YARN daemons, the HBase Thrift server, and so on. To do this, Data Explorer needs to have the certificate chains of these components' hosts in the Data Explorer trust store.

The Data Explorer truststore is a single PEM (Privacy Enhanced Mail) file that contains the certificate authority (CA) root certificate and all intermediate certificates to authenticate the certificate installed on each TLS/SSL-enabled server. These servers host the services with which Data Explorer communicates.

Note: A certificate is specific to a host. It is signed by a CA and tells the requesting client, which is Data Explorer in this case, that the host is the same one as is represented by the host public key. Data Explorer uses a chain of signing authority in its truststore to validate the CA that signed the host certificate.

Enabling Cloudera Data Explorer (Hue) as a TLS/SSL client

After you create a Data Explorer truststore file in PEM format, you can configure Data Explorer as a TLS/SSL client by using Cloudera Manager.

Procedure

1. Log in to Cloudera Manager as an Administrator.
2. Go to Clusters Hue service Configuration Hue TLS/SSL Server CA Certificate (PEM Format) ssl_cacerts and add the path to the *HUE_TRUSTSTORE*.pem file on the host that is running the Data Explorer web server.
3. Click Save Changes.
4. Restart the Data Explorer service.

Configuring Cloudera Data Explorer (Hue) as a TLS/SSL server

Data Explorer and other Python-based services expect certificates and keys to be stored in PEM (Privacy Enhanced Mail) format.

Before you enable TLS/SSL for the Data Explorer server, you must generate a private key and certificate by using the openssl command-line tool and reuse a host's existing Java keystore by converting it to the PEM format.

Enabling Cloudera Data Explorer (Hue) as a TLS/SSL server using Cloudera Manager

You can use Cloudera Manager to enable TLS/SSL for the Data Explorer server.

Procedure

1. Log in to Cloudera Manager as an Administrator.
2. Go to Clusters Hue service Configuration and filter by SCOPE Hue Server and CATEGORY Security .
3. Edit the following Data Explorer TLS/SSL properties according to your cluster configuration:
   - Enable TLS/SSL for Hue: Select the check box to encrypt communication between clients and Data Explorer with TLS/SSL.
   - Hue TLS/SSL Server Certificate File (PEM Format) ssl_certificate: Specifies the path to the TLS/SSL certificate on the host that is running the Data Explorer web server.

     Ensure that you include the complete chain in the ssl_certificate PEM file.

     The order of the certificates should be as follows from the top to bottom: server, intermediate, root.

     If there are multiple intermediate CA certificates, then you must add them in the correct order. For example:

```
Subject: CN=Hue Server Certificate
Issuer: CN=Intermediate 2

  Subject: CN=Intermediate 2
  Issuer: CN=Intermediate 1

    Subject: CN=Intermediate 1
    Issuer: CN=RootCA

      Subject: CN=RootCA
      Issuer: CN=RootCA
```

- Hue TLS/SSL Server Private Key File (PEM Format) ssl_private_key: Specifies the path to the TLS/SSL private key on the host running the Data Explorer web server.
- Hue TLS/SSL Private Key Password ssl_password: Specifies the password for the private key in the Data Explorer TLS/SSL Server Certificate and Private Key file.
- Hue TLS/SSL Server CA Certificate (PEM Format) ssl_cacerts: Specifies the path to the TLS/SSL certificate authority root certificate on the host that is running the Data Explorer web server.

4. Add the path to the certificate chain PEM file in [desktop] section of the Hue Service Advanced Configuration Snippet (Safety Valve) for hue_safety_valve.ini field:

```
[desktop]
ssl_certificate_chain=/[***PATH***]/[***TO***]/[***FULL-CHAIN***].pem
```

5. Click Save Changes.
6. Select Actions Restart to restart the Data Explorer service.

What to do next

Change the permissions for Data Explorer to read the certificates after you have enabled TLS/SSL as follows:

```
chmod 644 [***PATH-WHERE-SSL-FILES-FOR-HUE-ARE-LOCATED***]
```

Enabling TLS/SSL for Cloudera Data Explorer (Hue) Load Balancer

To configure the Data Explorer Load Balancer to use HTTPS or operate as a TLS/SSL server, you need a self-signed SSL certificate and a private key file. If the private key file is password protected, then you must configure the Data Explorer Load Balancer to use the corresponding key password.

Procedure

1. Log in to Cloudera Manager as an Administrator.
2. Go to Clusters Hue service Configuration Scope Load Balancer .
3. Enter the location of the file that contains the server certificate key for TLS/SSL on the host running Data Explorer Load Balancer in the Hue Load Balancer TLS/SSL Server Certificate File (PEM Format) field.

   The certificate file must be in the Privacy-Enhanced Mail (PEM) format.
4. Enter the location of the TLS/SSL file that contains the private key used for TLS/SSL on the host running Data Explorer Load Balancer, in the Hue Load Balancer TLS/SSL Server Private Key File (PEM Format) field.

   The certificate file must be in PEM format.
5. (Optional) If the private key file is password protected perform the following steps:
   a) Create a password file in your chosen security directory and insert the private key password as shown in the following example:

   ```
   echo "abc123" > /etc/security/password.txt
   ```

   Where abc123 is the private key password and password.txt is the password file.
   b) Set the file ownership and permissions as shown in the following example:

   ```
   chown hue:hue password.txt
   chmod 700 password.txt
   ```

   c) Enter the path to the file containing the passphrase used to encrypt the private key of the Data Explorer Load Balancer server in the Hue Load Balancer TLS/SSL Server SSLPassPhraseDialog field.
6. Click Save Changes.
7. Restart the Data Explorer service.

Enabling TLS/SSL communication with HiveServer2

For Data Explorer to communicate with HiveServer2 using TLS/SSL, Data Explorer needs the Hive certificate and certificate chain.

To enable TLS/SSL communication with HiveServer2, add the following properties in the [beeswax] section under [[ssl]] in the Cloudera Manager Hue Service Advanced Configuration Snippet (Safety Valve) for hue_safety_valve.ini configuration property:

Property	Description

`[beeswax]`
`[[ssl]]`
`enabled`
	Valid values: true | false
	Enables or disables TLS/SSL communication for this server.
	Default setting: false
	Example: enabled=true

`[beeswax]`
`[[ssl]]`
`cacerts`
	Valid values: directory path
	Specifies the path to the Certificate Authority certificates.
	Default setting: /etc/hue/cacerts.pem
	Example: cacerts=/opt/cloudera/security/CAcerts/cacerts

`[beeswax]`
`[[ssl]]`
`validate`
	Valid values: true | false
	Specifies whether Data Explorer validates certificates received from the server.
	Default setting: true
	Example: validate=true

Enabling TLS/SSL communication with Impala

For Data Explorer to communicate with Impala using TLS/SSL, Data Explorer needs the Impala certificate and certificate chain.

To enable TLS/SSL communication with Impala, add the following properties in the [impala] section under [[ssl]] in the Cloudera Manager Hue Service Advanced Configuration Snippet (Safety Valve) for hue_safety_valve.ini configuration property:

Property	Description

`[impala]`
`[[ssl]]`
`enabled`
	Valid values: true | false
	Enables or disables TLS/SSL communication for this server.
	Default setting: false
	Example: enabled=true

`[impala]`
`[[ssl]]`
`cacerts`
	Valid values: directory path
	Specifies the path to the Certificate Authority certificates.
	Default setting: /etc/hue/cacerts.pem
	Example: cacerts=/opt/cloudera/security/CAcerts/cacerts

`[impala]`
`[[ssl]]`
`validate`
	Valid values: true | false
	Specifies whether Data Explorer validates certificates received from the server.
	Default setting: true
	Example: validate=true

Securing database connections with TLS/SSL

Data Explorer uses different clients to communicate with each database internally. Client-specific options, such as secure connectivity can be configured using Cloudera Manager.

Procedure

1. Log in to Cloudera Manager as an administrator.
2. Go to Clusters Hue service Configuration and add the following section in the Hue Service Advanced Configuration Snippet (Safety Valve) for hue_safety_valve.ini field:

```
[desktop]
  [[database]]
    …
    options={"ssl":{"ca":"/tmp/ca-cert.pem"}}
```

This identifies the Certificate Authority (CA) certificate for the backend database. You can also identify public and private keys as follows:

```
options='{"ssl": {"ca": "/tmp/newcerts2/ca.pem", "key": "/tmp/newcerts2/
client-key.pem", "cert": “/tmp/newcerts2/client-cert.pem"}}'
```

3. Click Save Changes.
4. Restart the Data Explorer service.

Configuring Impala TLS/SSL

To protect sensitive information being transmitted, Impala supports TLS/SSL network encryption, between Impala and client programs, and between the Impala-related daemons running on different nodes in the cluster.

To configure Impala to listen for Beeswax and HiveServer2 requests on TLS/SSL-secured ports:

1. In Cloudera Manager, select the Impala service from the Clusters drop down.
2. In the Configuration tab, select ScopeImpala (Service-Wide).
3. Select CategorySecurity.
4. Edit the following property fields:

Property	Description
Enable TLS/SSL for Impala	Encrypt communication between clients (like ODBC, JDBC, and the Impala shell) and the Impala daemon using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
Impala TLS/SSL Server Certificate File (PEM Format)	Local path to the X509 certificate that identifies the Impala daemon to clients during TLS/SSL connections. This file must be in PEM format.
Impala TLS/SSL Server Private Key File (PEM Format)	Local path to the private key that matches the certificate specified in the Certificate for Clients. This file must be in PEM format.
Impala TLS/SSL Private Key Password	The password for the private key in the Impala TLS/SSL Server Certificate and Private Key file. If left blank, the private key is not protected by a password.
Impala TLS/SSL CA Certificate	The location on disk of the certificate, in PEM format, used to confirm the authenticity of SSL/TLS servers that the Impala daemons might connect to. Because the Impala daemons connect to each other, this should also include the CA certificate used to sign all the SSL/TLS Certificates. Without this parameter, SSL/TLS between Impala daemons will not be enabled.

Using TLS/SSL with Business Intelligence Tools

You can use Kerberos authentication, TLS/SSL encryption, or both to secure connections from JDBC and ODBC applications to Impala.

Configuring TLS/SSL Communication for the Impala Shell

Typically, a client program has corresponding configuration properties in Cloudera Manager to verify that it is connecting to the right server. For example, with SSL enabled for Impala, you use the following options when starting the `impala-shell` interpreter:

- --ssl: enables TLS/SSL for `impala-shell`.
- --ca_cert: the local pathname pointing to the third-party CA certificate, or to a copy of the server certificate for self-signed server certificates.

If --ca_cert is not set, `impala-shell` enables TLS/SSL, but does not validate the server certificate. This is useful for connecting to a known-good Impala that is only running over TLS/SSL, when a copy of the certificate is not available (such as when debugging customer installations).

For `impala-shell` to successfully connect to an Impala cluster that has the minimum allowed TLS/SSL version set to 1.2 (--ssl_minimum_version=tlsv1.2), the cluster that `impala-shell` runs on must have Python version 2.7.9 or higher (or a vendor-provided Python version with the required support. Some vendors patched Python 2.7.5 versions on Red Hat Enterprise Linux 7 and derivatives).

Specifying TLS/SSL Minimum Allowed Version and Ciphers

Depending on your cluster configuration and the security practices in your organization, you might need to restrict the allowed versions of TLS/SSL used by Impala. Older TLS/SSL versions might have vulnerabilities or lack certain features. You can use startup options for the `impalad`, `catalogd`, and `statestored` daemons to specify a minimum allowed version of TLS/SSL.

Add the following parameter into Cloudera Manager Impala Configuration . Search "Impala Command Line Argument Advanced Configuration Snippet (Safety Valve)"

```
--ssl_minimum_version=tlsv1.2
```

Along with specifying the version, you can also specify the allowed set of TLS ciphers by using the --ssl_cipher_list configuration setting. The argument to this option is a list of keywords, separated by colons, commas, or spaces, and optionally including other notation. For example:

```
--ssl_cipher_list="DEFAULT:!aNULL:!eNULL:!LOW:!EXPORT:!SSLv2:!SSLv3:!TLS1"
```

By default, the cipher list is empty, and Impala uses the default cipher list for the underlying platform. See the output of `man ciphers` for the full set of keywords and notation allowed in the argument string.

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.
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.

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 6:

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.

Configuring TLS/SSL encryption for the Kafka Connect role

Kafka Connect roles inherit the TLS/SSL configuration of the parent Kafka service. If you are deploying Kafka Connect roles under a Kafka service that already has TLS/SSL enabled, Cloudera Manager will automatically enable TLS/SSL for Connect as well. If required however, you can manually enable or disable TLS/SSL.

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 7:

Cloudera Manager Property	Description
Enable TLS/SSL for Kafka Connect ssl_enabled	Encrypt communication between clients and Kafka Connect using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
Kafka Connect 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 Connect is acting as a TLS/SSL server.
Kafka Connect TLS/SSL Server JKS Keystore File Password ssl_server_keystore_password	The password for the Kafka Connect keystore file.
Kafka Connect 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 Connect is acting as a TLS/SSL server.
Kafka Connect 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 Connect 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 Connect TLS/SSL Trust Store Password ssl_client_truststore_password	The password for the Kafka Connect 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.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 service.

Results

TLS/SSL encryption is configured for the Kafka Connect role.

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.

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 8: 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,DEFAULT
```

Note: For more information on principal mapping rules, see Principal name mapping.

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

Results

Kafka brokers are configured for TLS/SSL authentication.

What to do next

Configure your clients for TLS/SSL authentication.

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.

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.

Inter-broker security

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

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

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

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

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

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

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

Securing a non-secure Kafka deployment without downtime

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

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

Configuring multiple listeners

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

About this task

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

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

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

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

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

Before you begin

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

Procedure

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

3. Configure listeners using an advanced configuration snippet:

   Repeat the following steps for each Kafka broker role.

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

      For example:

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

      This example shows a configuration where the broker is accepting both SASL_SSL and SSL requests.

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

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

Results

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

Configuring TLS/SSL encryption manually for Apache Knox

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 Apache Knox.

Before you begin

- Review certificate requirements. See *TLS/SSL certificate requirements and recommendations* for more information.
- Review *Understanding Keystores and Truststores*.
- Create certificates and configure Cloudera Manager properties. See *Manually Configuring TLS Encryption for Cloudera Manager* for more information. Configuring TLS Encryption for Cloudera Manager Admin Console is required prior to enabling TLS encryption for Knox.

Note: To enhance security, it is recommended to disable the weak protocol SSLv2Hello in your environment. Follow the steps in *Disable weak protocol for Knox* to disable SSLv2Hello.

Procedure

1. From the Cloudera Manager site, go to Clusters > Knox.
2. Click the Configuration tab.
3. Enter tls in the search field. The security properties appear.
4. Edit the security properties according to the cluster configuration. For a list of security properties, see the Security section in *Knox Properties in Cloudera Runtime*.
5. Click Save Changes.
6. Restart the Knox service.

Related Information

TLS/SSL certificate requirements and recommendations
Understanding Keystores and Truststores
Manually Configuring TLS Encryption for Cloudera Manager
Knox Properties in Cloudera on cloud 7.1

Knox Properties for TLS

Learn about the Knox properties that you need to configure for TLS.

Knox Properties for TLS

Property	Value	Description
Knox Gateway TLS/SSL Client Trust Store File	gateway.httpclient.truststore.path	The location on disk of the trust store, in .jks format, used to confirm the authenticity of TLS/SSL servers that Knox Gateway might connect to. This is used when Knox Gateway is the client in a TLS/SSL connection. 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.
Knox Gateway TLS/SSL Client Trust Store Password	NA	The password for the Knox Gateway TLS/SSL Certificate 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.
Enable TLS/SSL for Knox Gateway	knox.enableTLS	Encrypt communication between clients and Knox Gateway using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
Knox Gateway TLS/SSL Server JKS Keystore File Location	gateway.tls.keystore.path	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Knox Gateway is acting as a TLS/SSL server. The keystore must be in JKS format.
Knox Gateway TLS/SSL Server JKS Keystore File Password	ssl_server_keystore_password	The password for the Knox Gateway JKS keystore file.

Disable weak protocol for Knox

How to disable weak security protocol SSLv2Hello for Knox.

About this task

Depending on your cluster configuration and the security practices in your organization, you might need to restrict the allowed versions of TLS/SSL used by Knox. Older TLS/SSL versions, such as SSLv2Hello, might have vulnerabilities or lack certain features.

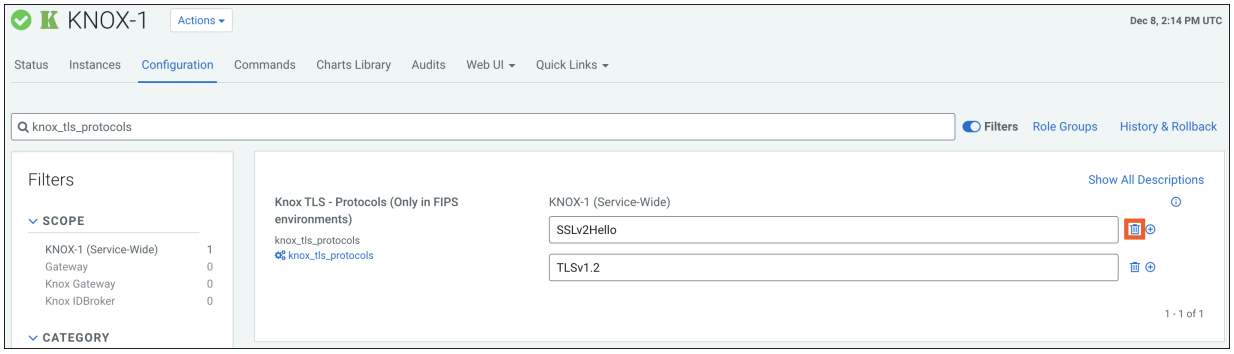
Before you begin

Your environment must support TLS 1.2 before removing SSLv2Hello in order to avoid compatibility issues.

Procedure

1. In Cloudera Manager, select the Knox service.
2. Go to Configuration.
3. Find the Knox TLS - Protocols (Only in FIPS environments) configuration property.

4. Click the Trash icon next to the SSLv2Hello entry.



5. Click the Save Changes(CTRL+S) button.
6. Refresh the Knox instances configuration by clicking the Stale Configuration: Refresh needed indicator and wait until the refresh process completes.
7. Validate using the Knox homepage.

Configuring TLS/SSL encryption for Kudu using Cloudera Manager

TLS/SSL encryption is enabled between Kudu servers and clients by default. You can enable TLS/SSL encryption for Kudu web UIs or configure the encryption using Cloudera Manager.

Procedure

1. In Cloudera Manager, navigate to Kudu Configuration .
2. In the Search field, type TLS/SSL to show the relevant properties.
3. Edit the following properties according to your cluster configuration:

Table 9: TLS/SSL Kudu properties

Property	Description
Master TLS/SSL Server Private Key File (PEM Format)	Set to the path containing the Kudu master host's private key (Privacy Enhanced Mail (PEM)-format). This is used to enable TLS/SSL encryption (over HTTPS) for browser-based connections to the Kudu master web UI.
Tablet Server TLS/SSL Server Private Key File (PEM Format)	Set to the path containing the Kudu tablet server host's private key (PEM-format). This is used to enable TLS/SSL encryption (over HTTPS) for browser-based connections to Kudu tablet server web UIs.
Master TLS/SSL Server Certificate File (PEM Format)	Set to the path containing the signed certificate (PEM-format) for the Kudu master host's private key (set in Master TLS/SSL Server Private Key File). The certificate file can be created by concatenating all the appropriate root and intermediate certificates required to verify trust.
Tablet Server TLS/SSL Server Certificate File (PEM Format)	Set to the path containing the signed certificate (PEM-format) for the Kudu tablet server host's private key (set in Tablet Server TLS/SSL Server Private Key File). The certificate file can be created by concatenating all the appropriate root and intermediate certificates required to verify trust.
Enable TLS/SSL for Master Server	Enables HTTPS encryption on the Kudu master web UI.
Enable TLS/SSL for Tablet Server	Enables HTTPS encryption on the Kudu tablet server web UIs.
Enable Secure Authentication, Encryption, and Web UI	Enables Kerberos for Kudu servers, clients and web servers, and enables encryption for Kudu servers and clients.

4. Click Save Changes.

Configure Lily HBase Indexer to use TLS/SSL

Although Cloudera recommends using AutoTLS, you also have the option to set up TLS manually for the Lily HBase Indexer.

About this task

To configure and enable Hadoop TLS/SSL for the Lily HBase Indexer (Key-Value Store Indexer) perform the following steps.

Procedure

1. Open the Cloudera Manager Admin Console and go to the Key-Value Store Indexer.
2. Click the Configuration tab.
3. Select Scope All .
4. Select Category All .
5. In the Search field, type TLS/SSL to show the Solr TLS/SSL properties.
6. Edit the following TLS/SSL properties according to your cluster configuration.

Note: These values must be the same for all hosts running the Key-Value Store Indexer role.

Table 10: Key-Value Store TLS/SSL Properties

Property	Description
HBase Indexer TLS/SSL Certificate Trust Store File	The location on disk of the truststore, in .jks format, used to confirm the authenticity of TLS/SSL servers that HBase Indexer might connect to. This is used when HBase Indexer is the client in a TLS/SSL connection. This truststore must contain the certificate(s) used to sign the service(s) being connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
HBase Indexer TLS/SSL Certificate Trust Store Password (Optional)	The password for the HBase Indexer TLS/SSL Certificate Trust Store 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 truststores are certificates, and certificates are public information.

7. Restart the service.

Configuring TLS/SSL encryption manually for Livy

You can enable TLS manually for the Apache Livy Server.

Procedure

1. In Cloudera Manager, select the Livy service from the Clusters drop-down menu.
2. In the Configuration tab, enter tls into the search box.
3. Edit the following property fields as needed for your cluster and environment:

Property	Description
Gateway TLS/SSL Trust Store File	The location on disk of the trust store, in .jks format, used to confirm the authenticity of TLS/SSL servers that Gateway 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.

Property	Description
Gateway TLS/SSL Trust Store Password	The password for the Gateway 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.
Enable TLS/SSL for Livy Server	Encrypt communication between clients and Livy Server using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
Livy Server 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 Livy Server is acting as a TLS/SSL server. The keystore must be in JKS format.
Livy Server TLS/SSL Server JKS Keystore File Password	The password for the Livy Server JKS keystore file.

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

Configuring TLS/SSL manually

If you use your own enterprise-generated certificates, you would need to manually configure TLS.

TLS/SSL certificate requirements and recommendations

If you use your own enterprise-generated certificates, you would need to manually configure TLS.

Before you manually configure TLS, ensure that the certificate that you use meets the following requirements.

Certificate requirements

Verify the following minimum requirements:

- 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 NiFi cluster node must use the same KeyStore password and key/certificate password. Ambari and Cloudera Manager do not support defining unique passwords per NiFi 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. If NiFi or NiFi Registry is behind Knox, do not use wildcard certificates for Knox.
- 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 for the NiFi service, 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/SSL encryption manually for NiFi and NiFi 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 NiFi and NiFi Registry.

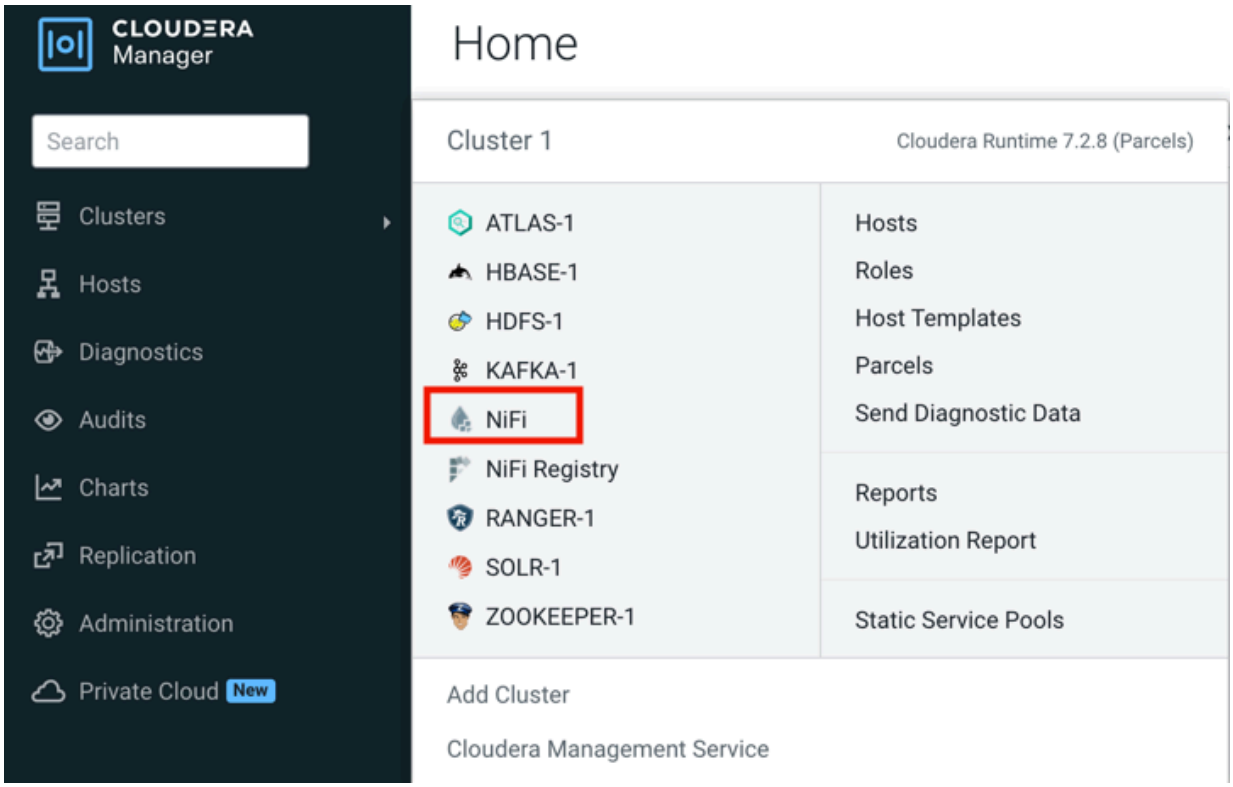
Before you begin

Ensure you have set up TLS for Cloudera Manager:

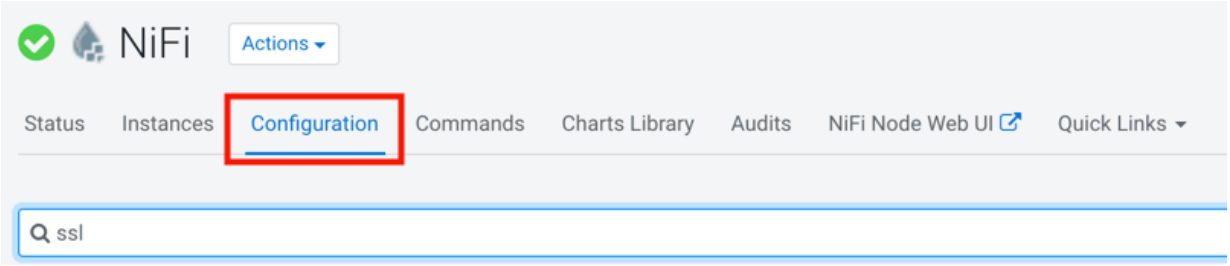
1. Review the requirements and recommendations for the certificates. For more information, see *TLS certificate requirements and recommendations*.
2. Generate the TLS certificates and configure Cloudera Manager. For more information, see *Manually configuring TLS for Cloudera Manager*.

Procedure

1. From Cloudera Manager, click Cluster NiFi .



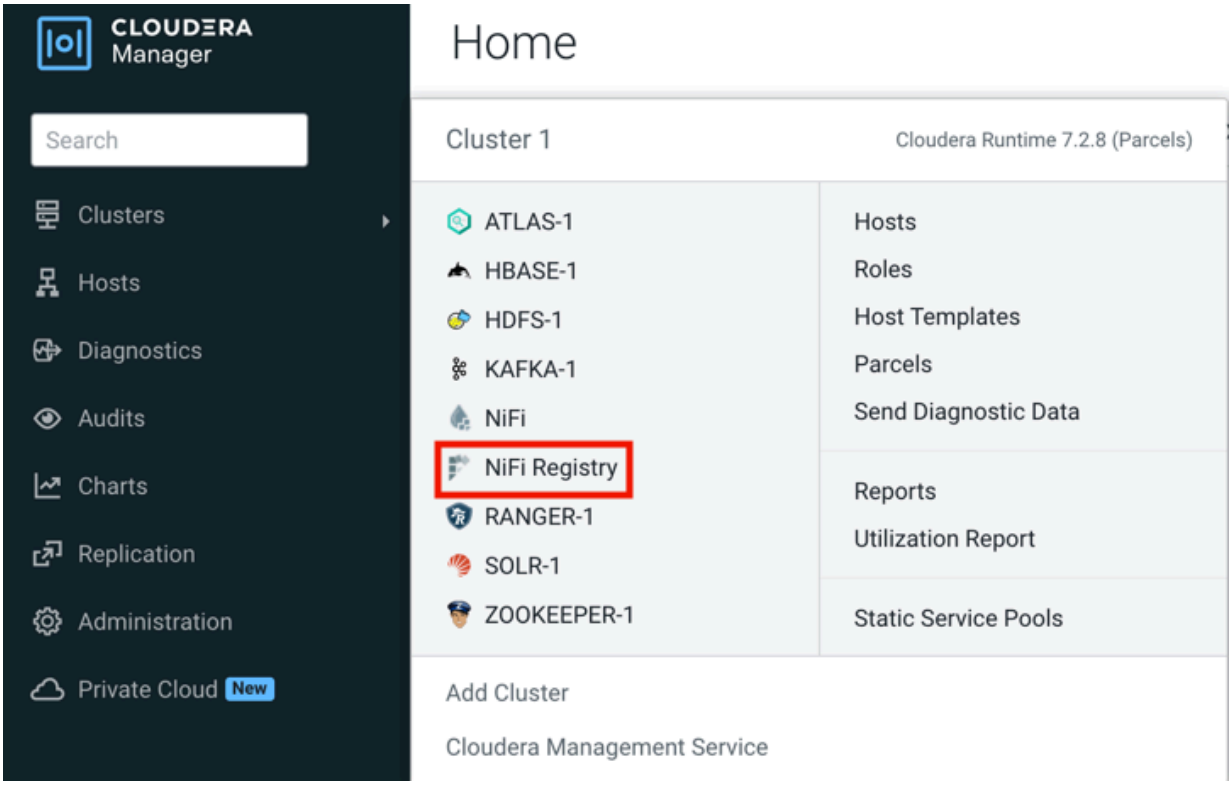
2. Click the **Configuration** tab.



3. Enter ssl in the Search field.

   The TLS/SSL Security properties for NiFi appear.
4. Edit the TLS/SSL Security properties.
5. Click Save Changes.
6. Restart the NiFi service.
7. Click Cluster NiFi Registry and repeat these steps to configure the TLS/SSL Security properties for NiFi Registry.

   If a property is not exposed in Cloudera Manager, use a safety valve to override the associated value.



NiFi TLS/SSL properties

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

The following table lists the TLS/SSL security properties for NiFi:

Property	Description
NiFi Node TLS/SSL Server JKS Keystore File Location nifi.security.keystore	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when NiFi Node is acting as a TLS/SSL server. The keystore must be in JKS format.

Property	Description
NiFi Node TLS/SSL Server JKS Keystore Type Passwordnifi.security.keystoreType	The type of the NiFi Node JKS keystore. It must be PKCS12 or JKS or BCFKS. JKS is the preferred type, BCFKS and PKCS12 files are loaded with BouncyCastle provider.
NiFi Node TLS/SSL Server JKS Keystore File Password nifi.security.keystorePasswd	The password for the NiFi Node JKS keystore file.
NiFi Node TLS/SSL Server JKS Keystore Key Password nifi.security.keyPasswd	The password that protects the private key contained in the JKS keystore used when NiFi Node is acting as a TLS/SSL server.
NiFi Node TLS/SSL Client Trust Store File nifi.security.truststore	The location on disk of the trust store, in JKS format, used to confirm the authenticity of TLS/SSL servers that NiFi Node might connect to. This is used when NiFi Node is the client in a TLS/SSL connection. 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.
NiFi Node TLS/SSL Client Trust Store Type nifi.security.truststoreType	The type of the NiFi Node TLS/SSL Certificate Trust Store. It must be PKCS12 or JKS or BCFKS. JKS is the preferred type, BCFKS and PKCS12 files are loaded with BouncyCastle provider.
NiFi Node TLS/SSL Client Trust Store Password nifi.security.truststorePasswd	The password for the NiFi Node TLS/SSL Certificate Trust Store File. This password is not required to access the trust store, the 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.

Note: Make sure to fill in all properties or NiFi will not start.

NiFi Registry TLS/SSL properties

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

The following table lists the TLS/SSL security properties for NiFi Registry:

Property	Description
NiFi Registry TLS/SSL Server JKS Keystore File Location nifi.registry.security.keystore	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when NiFi Registry is acting as a TLS/SSL server. The keystore must be in JKS format.
NiFi Registry TLS/SSL Server JKS Keystore Type Password nifi.registry.security.keystoreType	The type of the NiFi Registry JKS keystore. It must be PKCS12 or JKS or BCFKS. JKS is the preferred type, BCFKS and PKCS12 files are loaded with BouncyCastle provider.
NiFi Registry TLS/SSL Server JKS Keystore File Password nifi.registry.security.keystorePasswd	The password for the NiFi Registry JKS keystore file.
NiFi Registry TLS/SSL Server JKS Keystore Key Password nifi.registry.security.keyPasswd	The password that protects the private key contained in the JKS keystore used when NiFi Registry is acting as a TLS/SSL server.
NiFi Registry TLS/SSL Client Trust Store File nifi.registry.security.truststore	The location on disk of the trust store, in JKS format, used to confirm the authenticity of TLS/SSL servers that NiFi Registry might connect to. This is used when NiFi Registry is the client in a TLS/SSL connection. 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.
NiFi Registry TLS/SSL Client Trust Store Type nifi.registry.security.truststoreType	The type of the NiFi Registry TLS/SSL Certificate Trust Store. It must be PKCS12 or JKS or BCFKS. JKS is the preferred type, BCFKS and PKCS12 files are loaded with BouncyCastle provider.

Property	Description
NiFi Registry TLS/SSL Client Trust Store Password nifi.registry.security.truststorePasswd	The password for the NiFi Registry TLS/SSL Certificate 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.
NiFi Registry TLS/SSL Client Authentication nifi.registry.security.needClientAuth	This specifies that connecting clients must authenticate with a client cert. The default value is true. Setting the property to false will specify that connecting clients may optionally authenticate with a client cert, but may also login with a username and password against a configured identity provider.

Note: Make sure to fill in all properties or NiFi Registry will not start.

Configure TLS/SSL for Oozie

You can edit properties to enable TLS/SSL for Oozie, specify the keystore file location on the local file system, and set the password for the keystore.

Procedure

1. In Cloudera Manager, select the Oozie service.
2. Click the Configuration tab.
3. In the Search field, type TLS/SSL to show the Oozie TLS/SSL properties.
4. Edit the following TLS/SSL properties according to your cluster configuration.

Property	Description
Enable TLS/SSL for Oozie	Check this field to enable TLS/SSL for Oozie.
Oozie TLS/SSL Server JKS Keystore File Location	Path to the keystore file on the local file system.
Oozie TLS/SSL Server JKS Keystore File Password	Password for the keystore file.
Oozie TLS/SSL Client Trust Store File	Path to the client truststore file.
Oozie TLS/SSL Client Trust Store Password	Password for the truststore file.

5. If SSL is enabled for ZooKeeper, edit the following SSL properties:

Property	Description
Oozie ZooKeeper TLS/SSL Server JKS Keystore File Location	Path to the keystore file.
Oozie ZooKeeper TLS/SSL Server JKS Keystore File Password	Password for the keystore file.
Oozie ZooKeeper TLS/SSL Client Trust Store File	Path to the client truststore file.
Oozie ZooKeeper TLS/SSL Client Trust Store Password	Password for the truststore file.

6. Optionally, you can modify the values of the following properties:
   - Enabled TLS Protocols - List of Cipher Suite names that should be excluded.
   - Excluded Cipher Suites - TLS protocols accepted by the Oozie Server.
7. Click Save Changes.
8. Restart the Oozie service.

Configure TLS encryption manually for Phoenix Query Server

You can encrypt communication between clients and the Phoenix Query Server using Transport Layer Security (TLS) formerly known as Secure Socket Layer (SSL). You must follow these steps to manually configure TLS for Phoenix Query Server.

Before you begin

- Keystores containing certificates bound to the appropriate domain names must be accessible on all hosts running the Phoenix Query Server role of the Phoenix service.
- Keystores for Phoenix must be owned by the phoenix group, and have 0440 file permissions (that is, the file must be readable by the owner and group).
- Absolute paths to the keystore and truststore files must be specified. These settings apply to all hosts on which daemon roles of the Phoenix service run. Therefore, the paths you choose must be valid on all hosts.
- The Cloudera Manager version must support the TLS/SSL configuration for Phoenix at the service level. Ensure you specify absolute paths to the keystore and truststore files. These settings apply to all hosts on which daemon roles of the service in question run. Therefore, the paths you choose must be valid on all hosts.

  An implication of this is that the keystore file names for a given service must be the same on all hosts. If, for example, you have obtained separate certificates for Phoenix daemons on hosts node1.example.com and node2.example.com, you might have chosen to store these certificates in files called phoenix-node1.keystore and phoenix-node2.keystore (respectively). When deploying these keystores, you must give them both the same name on the target host — for example, phoenix.keystore.

Procedure

1. In Cloudera Manager, select the Phoenix service.
2. Click the Configuration tab.
3. Use the Scope / Query Server filter.
4. Search for tls.
5. Select Enable TLS/SSL for Query Server.
6. Edit the following TLS/SSL properties according to your configuration.

Table 11:

Property	Description
Query Server 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 Query Server is acting as a TLS/SSL server. The keystore must be in JKS format.
Query Server TLS/SSL Server JKS Keystore File Password	The password for the Query Server JKS keystore file.
Query Server TLS/SSL Client Trust Store File	The location on disk of the truststore file, in .jks format, used to confirm the authenticity of TLS/SSL servers to which the Query Server might connect. This is used when Query Server is the client in a TLS/SSL connection. This truststore file must contain the certificate(s) used to sign the connected service(s). If this parameter is not specified, the default list of known certificate authorities is used instead.
Query Server TLS/SSL Client Trust Store Password	The password for the Query Server TLS/SSL Certificate Trust Store File. This password is not mandatory to access the truststore; this field is optional. This password provides optional integrity checking of the file. The contents of truststores are certificates, and certificates are public information.

7. Click Save Changes.
8. Restart the Phoenix service.

Configure TLS/SSL encryption manually for Apache Ranger

How to manually configure TLS/SSL encryption for Apache Ranger.

About this task

Use this procedure when you want to manage your TLS/SSL certificates manually.

Procedure

1. In Cloudera Manager, select Ranger, then click the Configuration tab.
2. Under Category, select Security.
3. Set the following properties:

Note: Ranger supports the following keystore formats:

- JKS
- BCFKS in a FIPS-enabled cluster.

Table 12: Apache Ranger TLS/SSL Settings

Configuration Property	Description
Enable TLS/SSL for Ranger Admin ranger.service.https.attrib.ssl.enabled	Select this option to encrypt communication between clients and Ranger Admin using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
Ranger Admin TLS/SSL Server JKS Keystore File Location ranger.https.attrib.keystore.file	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Ranger Admin is acting as a TLS/SSL server. The keystore must be in JKS or BCFKS format.
Ranger Admin TLS/SSL Server JKS Keystore File Password ranger.service.https.attrib.keystore.pass	The password for the Ranger Admin JKS keystore file.
Ranger Admin TLS/SSL Client Trust Store File ranger.truststore.file	The location on disk of the truststore used to confirm the authenticity of TLS/SSL servers that Ranger Admin might connect to. This is used when Ranger Admin is the client in a TLS/SSL connection. This truststore must contain the certificate(s) used to sign the connected service(s). If this parameter is not provided, the default list of known certificate authorities is used.
Ranger Admin TLS/SSL Client Trust Store Password ranger.truststore.password	The password for the Ranger Admin TLS/SSL Certificate truststore file. This password is not required to access the truststore; therefore, this field is optional. The contents of truststores are certificates, and certificates are public information. This password provides optional integrity checking of the file.
Enable TLS/SSL for Ranger Tagsync	Select this option to encrypt communication between clients and Ranger Tagsync using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
Ranger Tagsync TLS/SSL Server JKS Keystore File Location xasecure.policymgr.clientssl.keystore	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Ranger Tagsync is acting as a TLS/SSL server. The keystore must be in JKS or BCFKS format.
Ranger Tagsync TLS/SSL Server JKS Keystore File Password xasecure.policymgr.clientssl.keystore.password	The password for the Ranger Tagsync JKS keystore file.

Configuration Property	Description
Ranger Tagsync TLS/SSL Trust Store File xasecure.policymgr.clientssl.truststore	The location on disk of the trust store, in .jks format, used to confirm the authenticity of TLS/SSL servers that Ranger Tagsync might connect to. This trust store must contain the certificate(s) used to sign the service(s) connected to. Ranger Tagsync connects to Ranger Admin. If Ranger Admin is SSL enabled, make sure you add a Ranger Admin certificate in the trust store. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
Ranger Tagsync TLS/SSL Client Trust Store Password xasecure.policymgr.clientssl.truststore.password	The password for the Ranger Tagsync TLS/SSL Certificate truststore file. This password is not mandatory to access the truststore. It is used to check the integrity of the file; this field is optional. The contents of truststores are certificates, and certificates are public information.
Ranger Usersync TLS/SSL Client Trust Store File ranger.usersync.truststore.file	The location on disk of the truststore, in JKS format, used to confirm the authenticity of TLS/SSL servers that Ranger Usersync might connect to. This is used when Ranger Usersync is the client in a TLS/SSL connection. This truststore must contain the certificate(s) used to sign the connected service(s). Ranger Usersync connects to Ranger Admin to sync users into Ranger. If Ranger Admin is SSL enabled, make sure you add a Ranger Admin certificate in the trust store. If this parameter is not provided, the default list of known certificate authorities is used.
Ranger Usersync TLS/SSL Client Trust Store Password ranger.usersync.truststore.password	The password for the Ranger Usersync TLS/SSL certificate truststore file. This password is not required to access the truststore; this field is optional. This password provides optional integrity checking of the file. The contents of trust stores are certificates, and certificates are public information.

4. In Filters Search , type ranger.service.https.attrib.keystore.keyalias to set the Ranger Admin TLS/SSL Keystore File Alias property.

Table 13: Ranger Admin TLS/SSL Setting

Configuration Property	Description
Ranger Admin TLS/SSL Keystore File Alias ranger.service.https.attrib.keystore.keyalias	The alias used for the Ranger Admin TLS/SSL keystore file. If host FQDN is used as an alias while creating a keystore file, the default placeholder value {{RANGER_ADMIN_HOST}} is replaced with the host FQDN where Ranger Admin will be installed in the current cluster. The placeholder can be replaced to have a custom alias used while creating the keystore file. If using a custom alias which is the same as the host short name, use {{RANGER_ADMIN_HOST_UQDN}} placeholder as a value.

5. Click Save Changes.

Configure TLS/SSL encryption manually for Ranger KMS

How to manually configure TLS/SSL encryption for Ranger KMS

About this task

Procedure

1. In Cloudera Manager, select Ranger KMS, then click the Configuration tab.
2. Under Category, select Security.
3. Set the following properties:

Table 14: Ranger KMS TLS/SSL Settings

Configuration Property	Description
Enable TLS/SSL for Ranger KMS Server ranger.service.https.attrib.ssl.enabled	Encrypt communication between clients and Ranger KMS Server using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
Ranger KMS Server TLS/SSL Server JKS Keystore File Location ranger.service.https.attrib.keystore.file	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Ranger KMS Server is acting as a TLS/SSL server. The keystore must be in JKS format.
Ranger KMS Server TLS/SSL Server JKS Keystore File Password ranger.service.https.attrib.keystore.pass	The password for the Ranger KMS Server JKS keystore file.
Ranger KMS Server TLS/SSL Trust Store File xasecure.policymgr.clientssl.truststore	The location on disk of the trust store, in .jks format, used to confirm the authenticity of TLS/SSL servers that Ranger KMS Server might connect to. This trust store must contain the certificate(s) used to sign the service(s) connected to. The Ranger KMS plugin inside the Ranger KMS Server connects to Ranger Admin to download the authorization policies. If Ranger Admin is SSL enabled, make sure you add a Ranger Admin certificate in the trust store. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
Ranger KMS Server TLS/SSL Trust Store Password xasecure.policymgr.clientssl.truststore.password	The password for the Ranger KMS Server 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.

4. In Filters Search , type ranger.service.https.attrib.keystore.keyalias to set the Ranger KMS Server TLS/SSL Keystore File Alias property.

Table 15: Ranger KMS Server TLS/SSL Keystore Alias Property Settings

Configuration Property	Description
Ranger KMS Server TLS/SSL Keystore File Alias ranger.service.https.attrib.keystore.keyalias	The alias for the Ranger KMS Server TLS/SSL keystore file. If host FQDN is used as an alias while creating a keystore file, the {{HOST}} default placeholder value will be replaced with the host FQDN where Ranger KMS Server will be installed in the current cluster. The placeholder can be replaced to have a custom alias used while creating the keystore file. If using a custom alias which is the same as host short name then use {{HOST_UQDN}} placeholder as a value.

5. Click Save Changes.

Overriding custom keystore alias on a Ranger KMS Server

Use this procedure to override the custom keystore alias on a Ranger KMS server.

The custom keystore alias may need to be overridden in the following scenarios:

- User has manually enabled TLS/SSL during fresh installations of Ranger KMS, and the keystore alias was not added to the hostname.

Overriding custom keystore alias while configuring TLS/SSL on a single instance of Ranger KMS Server

1. In Cloudera Manager, select Ranger KMS Configuration , and search for ranger.service.https.attrib.keystore .keyalias to set the custom alias value for the Ranger KMS Server TLS/SSL Keystore File Alias configuration parameter.
2. Click Save Changes.
3. Restart the Ranger KMS service.

Overriding custom keystore alias while configuring TLS/SSL on multiple instances of Ranger KMS Server

1. In Cloudera Manager, select Ranger KMS Instances and select Ranger KMS Server role Configuration . Use the Add (+) icons for the Ranger KMS Server Advanced Configuration Snippet (Safety valve) for conf/ranger-kms-site.xml property to add the following property:

   ```
   ranger.service.https.attrib.keystore.keyalias = <expected alias>
   ```

   This overrides the configuration on the host on which the current Ranger KMS Server role is available.
2. Repeat Step 1 for all the other Ranger KMS Servers to override the configuration by using the Ranger KMS Server Advanced Configuration Snippet (Safety valve) for conf/ranger-kms-site.xml property.
3. Restart the Ranger KMS service.

Note: When high-availability has been enabled for Ranger KMS, the keystore may not have the same alias for different KMS instances. In such cases, use FQDN as the alias or add the custom key alias configuration in the Ranger KMS Server Advanced Configuration Snippet (Safety valve) for conf/ranger-kms-site.xml property of each host.

Configure TLS/SSL encryption manually for Ranger RMS

How to manually configure TLS/SSL encryption for Ranger RMS

About this task

Procedure

1. In Cloudera Manager, select Ranger KMS, then click the Configuration tab.
2. Under Category, select Security.
3. Set the following properties:

Table 16: Ranger RMS TLS/SSL Settings

Configuration Property	Description
Enable TLS/SSL for Ranger RMS Server ranger-rms.service.https.attrib.ssl.enabled	Encrypt communication between clients and Ranger RMS Server using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
Ranger RMS Server TLS/SSL Server JKS Keystore File Location ranger-rms.service.https.attrib.keystore.file	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Ranger RMS Server is acting as a TLS/SSL server. The keystore must be in JKS format.

Configuration Property	Description
Ranger RMS Server TLS/SSL Server JKS Keystore File Password ranger-rms.service.https.attrib.keystore.pass	The password for the Ranger RMS Server JKS keystore file.
Ranger RMS Server TLS/SSL Trust Store File ranger-rms.truststore.file	The location on disk of the trust store, in .jks format, used to confirm the authenticity of TLS/SSL servers that Ranger RMS Server 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.
Ranger RMS Server TLS/SSL Trust Store Password ranger-rms.truststore.password	The password for the Ranger RMS Server 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.

4. In Filters Search , type ranger-rms.service.https.attrib.keystore.keyalias to set the Ranger RMS Server TLS/SSL Keystore File Alias property.

Table 17: Ranger RMS Server TLS/SSL Keystore File Alias Settings

Configuration Property	Description
Ranger RMS Server TLS/SSL Keystore File Alias ranger-rms.service.https.attrib.keystore.keyalias	The alias for the Ranger RMS Server TLS/SSL keystore file. If host FQDN is used as an alias while creating a keystore file, the {{HOST}} default placeholder value will be replaced with the host FQDN where Ranger RMS Server will be installed in the current cluster. The placeholder can be replaced to have a custom alias used while creating the keystore file. If using a custom alias which is the same as host short name then use {{HOST_UQDN}} placeholder as a value.

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:

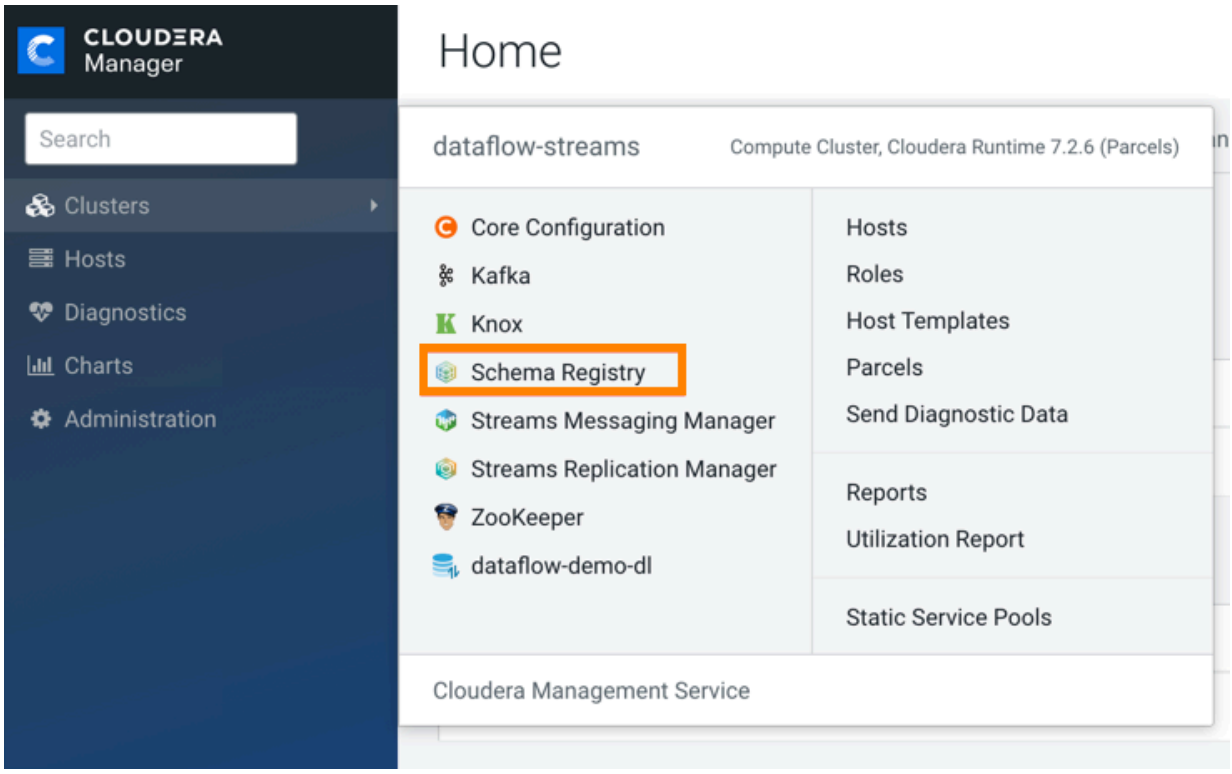
1. Review the requirements and recommendations for the certificates.

   For more information, see *TLS Certificate Requirements and Recommendations*.
2. Generate the TLS certificates and configure Cloudera Manager.

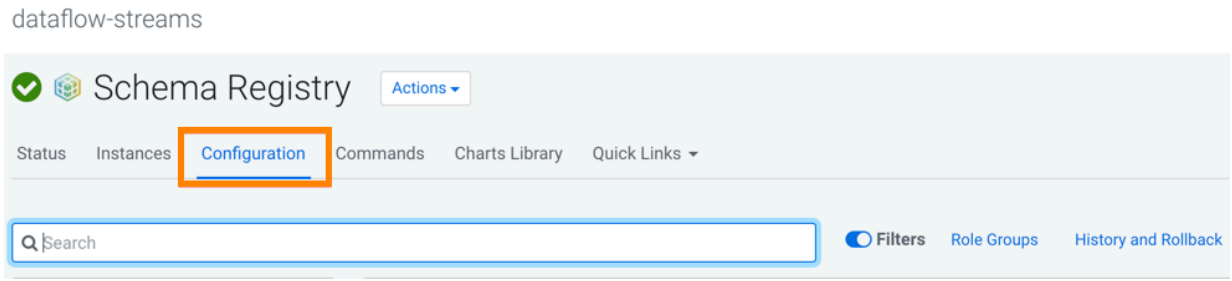
   For more information, see *Manually Configuring TLS Encryption for Cloudera Manager*.

Procedure

1. In Cloudera Manager, click Cluster Schema Registry .



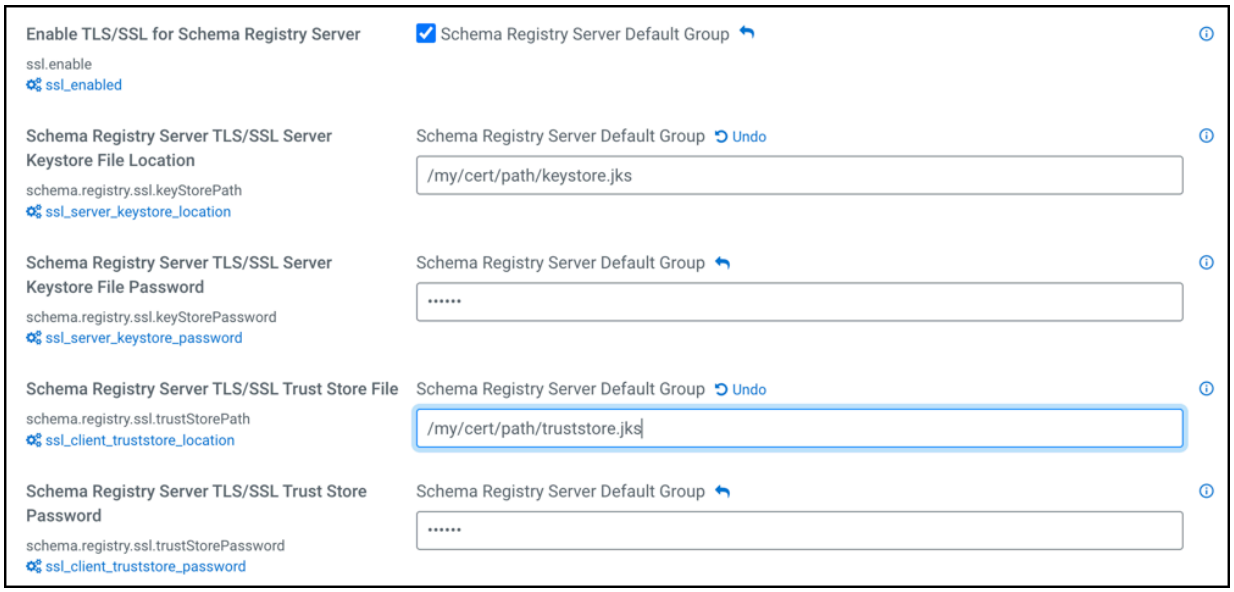
2. Click **Configuration**.



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.

Configure TLS/SSL encryption for Solr

Although Cloudera recommends using AutoTLS, you also have the option to set up TLS manually for Cloudera Search.

Before you begin

Minimum required role: Configurator (Also provided by Cluster Administrator, Full Administrator)

- The Solr service must be running.
- Keystores for Solr must be readable by the solr user. This could be a copy of the Hadoop services' keystore with permissions 0440 and owned by the solr group.
- Truststores must have permissions 0444 (that is, readable by all).
- Specify absolute paths to the keystore and truststore files. These settings apply to all hosts on which daemon roles of the Solr service run. Therefore, the paths you choose must be valid on all hosts.
- In case there is a DataNode and a Solr server running on the same host, they can use the same certificate.

For more information on obtaining signed certificates and creating keystores, see Encrypting Data in Transit. You can also view the upstream Solr documentation.

About this task

An additional consideration when configuring TLS/SSL for Solr HA is to allow clients to talk to Solr servers (the target servers) through the load balancer using TLS/SSL. To achieve this, you have to configure the load balancer for TLS/SSL pass-through, which means the load balancer does not perform encryption/decryption but simply passes traffic from clients and servers to the appropriate target host. See the documentation of your load balancer for details.

Procedure

1. Open the Cloudera Manager Admin Console and go to the Solr service.
2. Click the Configuration tab.
3. Select Scope All .

4. In the Search field, type TLS/SSL to show the Solr TLS/SSL properties.
5. Edit the following properties according to your cluster configuration.

Note: These values must be the same for all hosts running the Solr role.

Table 18: Solr TLS/SSL Properties

Property	Description
Enable TLS/SSL for Solr	Check this field to enable TLS for Solr.
Solr 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 Solr is acting as a TLS/SSL server. The keystore must be in JKS format.
Solr TLS/SSL Server JKS Keystore File Password	Password for the Solr JKS keystore.
Solr TLS/SSL Client Trust Store File	Required in case of self-signed or internal CA signed certificates. The location on disk of the truststore, in .jks format, used to confirm the authenticity of TLS/SSL servers that Solr might connect to. This is used when Solr is the client in a TLS/SSL connection. This truststore must contain the certificate(s) used to sign the service(s) being connected to. If this parameter is not provided, the default list of well-known certificate authorities is used instead.
Solr TLS/SSL Client Trust Store Password	The password for the Solr TLS/SSL Certificate Trust Store 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 truststores are certificates, and certificates are public information.

6. Enter a Reason for Change, and then click Save Changes to commit your changes.
7. Launch the Stale Configuration wizard to restart the Solr service and any dependent services.

What to do next

If Ranger authorization has been enabled for the Solr service, you need to update the Solr Collection URL (for a resource-based policy) or Solr URL (for a resource-based service) from http://host_ip:8983/solr to https://host_ip:8985/solr on the Ranger Admin Web UI.

Additional configuration steps when using a load balancer TLS/SSL for Solr HA

About this task

To configure a load balancer:

Procedure

1. Go to the Solr service.
2. Click the Configuration tab.
3. Select Scope Solr .
4. Enter the hostname and port number of the load balancer in the Solr Load Balancer property in the format host name:port number.

Note:

When you set this property, Cloudera Manager regenerates the keytabs for Solr roles. The principal in these keytabs contains the load balancer hostname.

If there are services that depend on this Solr service, such as Data Explorer, those services use the load balancer to communicate with Solr.

5. Enter a Reason for change, and then click Save Changes to commit the changes.
6. Restart Solr and any dependent services or restart the entire cluster for this configuration to take effect.

Configuring TLS/SSL encryption manually for Spark

You can enable TLS manually for the Spark History Server.

Procedure

1. In Cloudera Manager, select the Spark service from the Clusters drop-down menu.
2. In the Configuration tab, enter tls into the search box.
3. Edit the following property fields as needed for your cluster and environment:

Property	Description
TLS/SSL Protocol	The version of the TLS/SSL protocol to use when TLS/SSL is enabled.
Enabled SSL/TLS Algorithms	A comma-separated list of algorithm names to enable when TLS/SSL is enabled. By default, all algorithms supported by the JRE are enabled.
TLS/SSL Port Number	Port where to listen for TLS/SSL connections. HTTP connections will be redirected to this port when TLS/SSL is enabled.
Enable TLS/SSL for History Server	Encrypt communication between clients and History Server using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
History Server 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 History Server is acting as a TLS/SSL server. The keystore must be in JKS format.
History Server TLS/SSL Server JKS Keystore File Password	The password for the History Server JKS keystore file.

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

Enabling TLS/SSL for the Streams Replication Manager service

TLS/SSL can be enabled and configured for the Streams Replication Manager service (Driver and Service roles) with various configuration properties available in Cloudera Manager. Configuring these properties affects the security configuration of Streams Replication Manager in multiple ways.

About this task

Both the Driver and Service roles of Streams Replication Manager have a number of TLS/SSL related properties associated with them. A dedicated TLS/SSL feature toggle exists for both roles. These are the Enable TLS/SSL for SRM Driver and Enable TLS/SSL for SRM Service properties. In addition to the feature toggles, there are a number of other properties that can be used to configure key and truststore information.

Configuring the feature toggles and the key/truststore related properties have the following effects on Streams Replication Manager's security configuration:

- The Streams Replication Manager Service role's REST server becomes secured and uses HTTPS.
- The Streams Replication Manager Driver role's replication specific Connect REST servers become secured and use HTTPS. In addition, client authentication will also be required from any client connecting to these servers.

  Note: The replication specific Connect REST servers are for internal use only. They enable communication between Driver role instances. Third party clients should not interface with them.

- If the deployment has a co-located Kafka cluster and that cluster was configured using a service dependency, both the Service and Driver roles will use the keystore and truststore information when they establish a connection with the co-located Kafka cluster.

- Both the Driver and Service roles will use these properties as fallback configurations when establishing a connection to a Kafka cluster.

  That is, if there is a Kafka cluster in your configuration that has its protocol specified as SSL, but no trust or keystore information is set for it, the roles will use the truststore and keystore configured with these properties.

Important: Configuring the feature toggles and key/truststore properties on their own do not enable Streams Replication Manager to connect to or replicate a TLS/SSL enabled external Kafka cluster. They also do not have an effect on the srm-control tool's security configuration.

Configuring these properties is part of the process of defining and adding clusters described in *Defining and adding clusters for replication*. Depending on how you set up your co-located cluster, these properties might already be configured.

If you configured the co-located cluster with a service dependency, then these properties are configured in your deployment and no additional steps are needed to enable or configure TLS/SSL.

However, if you chose to set up the co-located cluster with a Kafka credential, these properties might not be configured. Although in a case like this the properties required by Streams Replication Manager to access the co-located cluster will be set, the server functionality of the roles will not be TLS/SSL enabled. Additionally, while not crucial, no fallback properties will be set up for security either. In a case like this Cloudera recommends that you complete the following steps.

Procedure

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

Table 19:

Cloudera Manager Property	Description
Enable TLS/SSL for SRM Driver	Encrypt communication between clients and Streams Replication Manager Driver using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
SRM Driver 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 Streams Replication Manager Driver is acting as a TLS/SSL server. The keystore must be in JKS format.
SRM Driver TLS/SSL Server JKS Keystore File Password	The password for the Streams Replication Manager Driver JKS keystore file.
SRM Driver TLS/SSL Server JKS Keystore Key Password	The password that protects the private key contained in the JKS keystore used when Streams Replication Manager Driver is acting as a TLS/SSL server.
SRM Driver TLS/SSL Trust Store File	The location on disk of the trust store, in .jks format, used to confirm the authenticity of TLS/SSL servers that Streams Replication Manager Driver 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.
SRM Driver TLS/SSL Trust Store Password	The password for the Streams Replication Manager Driver 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.
Enable TLS/SSL for SRM Service	Encrypt communication between clients and Streams Replication Manager Service using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL))

Cloudera Manager Property	Description
SRM Service 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 Streams Replication Manager Service is acting as a TLS/SSL server. The keystore must be in JKS format.
SRM Service TLS/SSL Server JKS Keystore File Password	The password for the Streams Replication Manager Service JKS keystore file.
SRM Service TLS/SSL Server JKS Keystore Key Password	The password that protects the private key contained in the JKS keystore used when Streams Replication Manager Service is acting as a TLS/SSL server.
SRM Service TLS/SSL Trust Store File	The location on disk of the trust store, in .jks format, used to confirm the authenticity of TLS/SSL servers that Streams Replication Manager Service 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.
SRM Service TLS/SSL Trust Store Password	The password for the Streams Replication Manager Service 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.

4. Click Save Changes.
5. Restart the Streams Replication Manager service.

Enabling TLS Encryption for Streams Messaging Manager on Cloudera on premises

Learn how to enable TLS/SSL encryption for Streams Messaging Manager on Cloudera on premises to secure the communication of sensitive information. You can enable the settings in Cloudera Manager according to the cluster configuration.

About this task

If Kerberos is enabled, then you must enable SSL for Streams Messaging Manager. Streams Messaging Manager UI fails to load if Kerberos is enabled and SSL is not enabled.

Also, if Kafka has Kerberos/SSL enabled, the same should be enabled for Streams Messaging Manager.

Procedure

1. Log in to Cloudera Manager.
2. Select the Streams Messaging Manager service.
3. Click Configuration from the menu bar.
4. In the Search field, type TLS/SSL to show the Streams Messaging Manager TLS/SSL properties.

   The security related properties appear.
5. Edit the security properties according to the cluster configuration.
6. Click Save Changes.

TLS/SSL settings for Streams Messaging Manager

To enable TLS/SSL settings for Streams Messaging Manager, you need to configure Streams Messaging Manager server properties, Streams Messaging Manager UI properties, and Streams Messaging Manager Server's Oracle TLS connection properties in Cloudera Manager according to the cluster configuration.

Table 20: TLS/SSL Settings for Streams Messaging Manager

Properties	Description
Streams Messaging Manager Server properties	
Enable TLS/SSL for Streams Messaging Manager Rest Admin Server ssl.enable	Encrypt communication between clients and Streams Messaging Manager Rest Admin Server using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
Streams Messaging Manager port (SSL) streams.messaging.manager.ssl.port	HTTPS port Streams Messaging Manager rest server runs on when SSL is enabled.
Streams Messaging Manager Admin Port (SSL) streams.messaging.manager.ssl.adminPort	HTTPS admin port Streams Messaging Manager rest server runs on when SSL is enabled.
SSL Keystore Type streams.messaging.manager.ssl.keyStoreType	The keystore type. Required if Streams Messaging Manager rest server's SSL is enabled. e.g. PKCS12 or JKS. If it is left empty then the keystore type will come from Cloudera Manager settings.
SSL TrustStore Type streams.messaging.manager.ssl.trustStoreType	The truststore type. Required if Streams Messaging Manager's ssl is enabled. e.g. PKCS12 or JKS. If it is left empty then the keystore type will come from Cloudera Manager settings.
Streams Messaging Manager Rest Admin Server TLS/SSL Server JKS Keystore File Location streams.messaging.manager.ssl.keyStorePath	The path to the TLS/SSL keystore file containing the server certificate and private key used for TLS/SSL. Used when Streams Messaging Manager Rest Admin Server is acting as a TLS/SSL server.
Streams Messaging Manager Rest Admin Server TLS/SSL Server Keystore File Password	The password for the Streams Messaging Manager Rest Admin Server keystore file.
Streams Messaging Manager Rest Admin Server TLS/SSL Server Keystore Key Password	The password that protects the private key contained in the keystore used when Streams Messaging Manager Rest Admin Server is acting as a TLS/SSL server.
Streams Messaging Manager Rest Admin Server TLS/SSL Trust Store File streams.messaging.manager.ssl.trustStorePath	The location on disk of the trust store used to confirm the authenticity of TLS/SSL servers that Streams Messaging Manager Rest Admin Server might connect to. This is used when Streams Messaging Manager Rest Admin Server is the client in a TLS/SSL connection. 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.
Streams Messaging Manager Rest Admin Server TLS/SSL Trust Store Password	The password for the Streams Messaging Manager Rest Admin Server TLS/SSL Certificate 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.
Cloudera Manager Metrics TrustStore Type cm.metrics.truststore.type	Cloudera Manager's truststore type. If it is left empty then the keystore type will come from Cloudera Manager settings. If it is left empty then the keystore type will come from Cloudera Manager settings.
SSL ValidateCerts streams.messaging.manager.ssl.validateCerts	Whether or not to validate TLS certificates before starting. If enabled, it will refuse to start with expired or otherwise invalid certificates.
SSL validatePeers streams.messaging.manager.ssl.validatePeers	Whether or not to validate TLS peer certificates.
Streams Messaging Manager UI properties	
Enable TLS/SSL for Streams Messaging Manager UI Server streams.messaging.manager.ui.ssl.enable	Encrypt communication between clients and Streams Messaging Manager UI Server using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).
Streams Messaging Manager UI Server TLS/SSL Server Private Key File (PEM Format) streams.messaging.manager.ui.ssl.private.key.location	The path to the TLS/SSL file containing the private key used for TLS/SSL. Used when Streams Messaging Manager UI Server is acting as a TLS/SSL server. The certificate file must be in PEM format.

Properties	Description
Streams Messaging Manager UI Server TLS/SSL Server Certificate File (PEM Format) streams.messaging.manager.ui.ssl.cert.location	The path to the TLS/SSL file containing the server certificate key used for TLS/SSL. Used when Streams Messaging Manager UI Server is acting as a TLS/SSL server. The certificate file must be in PEM format.
Streams Messaging Manager UI Server TLS/SSL Server CA Certificate (PEM Format) streams.messaging.manager.ui.ssl.ca.cert.location	The path to the TLS/SSL file containing the certificate of the certificate authority (CA) and any intermediate certificates used to sign the server certificate. Used when Streams Messaging Manager UI Server is acting as a TLS/SSL server. The certificate file must be in PEM format, and is usually created by concatenating all of the appropriate root and intermediate certificates.
Streams Messaging Manager UI Server TLS/SSL Private Key Password	The password for the private key in the Streams Messaging Manager UI Server TLS/SSL Server Certificate and Private Key file. If left blank, the private key is not protected by a password.
Streams Messaging Manager UI Server TLS/SSL Certificate Trust Store File streams.messaging.manager.ui.ssl.trust.store.location	The location on disk of the trust store, in .pem format, used to confirm the authenticity of TLS/SSL servers that Streams Messaging Manager UI Server might connect to. This is used when Streams Messaging Manager UI Server is the client in a TLS/SSL connection. 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.
Streams Messaging Manager Server's Oracle TLS connection properties	
Enable TLS with Oracle DB streams.messaging.manager.enable.TLS.Oracle	Enable TLS with Oracle as DB for Schema Registry.
Oracle.net.ssl_version streams.messaging.manager.oracle.net.ssl_version	Oracle net ssl version.
Oracle TLS javax.net.ssl.keyStore streams.messaging.manager.javax.net.ssl.keyStore	Path to keystore file if enabling TLS using Oracle DB.
Oracle TLS javax.net.ssl.keyStoreType streams.messaging.manager.javax.net.ssl.keyStoreType	KeyStoreType type if enabling TLS using Oracle DB.
Oracle TLS javax.net.ssl.keyStorePassword streams.messaging.manager.javax.net.ssl.keyStorePassword	KeyStorePassword if enabling TLS using Oracle DB.
Oracle TLS javax.net.ssl.trustStore streams.messaging.manager.javax.net.ssl.trustStore	Required Path to truststore file if enabling TLS using Oracle DB.
Oracle TLS javax.net.ssl.trustStoreType streams.messaging.manager.javax.net.ssl.trustStoreType	Required Truststore type if enabling TLS using Oracle DB.
Oracle TLS javax.net.ssl.trustStorePassword streams.messaging.manager.javax.net.ssl.trustStorePassword	TrustStorePassword type if enabling TLS using Oracle DB.
Oracle TLS oracle.net.ssl_cipher_suites streams.messaging.manager.oracle.net.ssl_cipher_suites	net ssl cipher suites if enabling TLS using Oracle DB e.g. SSL_DH_DSS_WITH_DES_CBC_SHA.
Oracle TLS oracle.net.ssl_server_dn_match streams.messaging.manager.oracle.net.ssl_server_dn_match	ssl server domain name match if enabling TLS using Oracle DB.
Oracle TLS oracle.net.authentication_services streams.messaging.manager.oracle.net.authentication_services	Oracle net authentication service if enabling TLS using Oracle DB.

Configuring TLS/SSL for Core Hadoop Services

TLS/SSL for the core Hadoop services (HDFS and YARN) must be enabled as a group.

TLS/SSL for the core Hadoop services (HDFS and YARN) must be enabled as a group. Enabling TLS/SSL on HDFS is required before it can be enabled on YARN.

Note: If you enable TLS/SSL for HDFS, you must also enable it for YARN.

The steps in the following topics include enabling Kerberos authentication for HTTP Web-Consoles. Enabling TLS/SSL for the core Hadoop services on a cluster without enabling authentication displays a warning.

Before You Begin

- Before enabling TLS/SSL, keystores containing certificates bound to the appropriate domain names will need to be accessible on all hosts on which at least one HDFS or YARN daemon role is running.
- Since HDFS and YARN daemons act as TLS/SSL clients as well as TLS/SSL servers, they must have access to truststores. In many cases, the most practical approach is to deploy truststores to all hosts in the cluster, as it may not be desirable to determine in advance the set of hosts on which clients will run.
- Keystores for HDFS and YARN must be owned by the hadoop group, and have permissions 0440 (that is, readable by owner and group). Truststores must have permissions 0444 (that is, readable by all)
- Cloudera Manager supports TLS/SSL configuration for HDFS and YARN at the service level. For each of these services, you must specify absolute paths to the keystore and truststore files. These settings apply to all hosts on which daemon roles of the service in question run. Therefore, the paths you choose must be valid on all hosts.

  An implication of this is that the keystore file names for a given service must be the same on all hosts. If, for example, you have obtained separate certificates for HDFS daemons on hosts node1.example.com and node2.example.com, you might have chosen to store these certificates in files called hdfs-node1.keystore and hdfs-node2.keystore (respectively). When deploying these keystores, you must give them both the same name on the target host — for example, hdfs.keystore.
- Multiple daemons running on a host can share a certificate. For example, in case there is a DataNode and an Oozie server running on the same host, they can use the same certificate.

Configuring TLS/SSL for HDFS

You must enable TLS/SSL HDFS properties, TLS/SSL Client TrustStore properties, and allow web UI authentication.

About this task

Cloudera recommends you enable web UI authentication for the HDFS service. Web UI authentication uses SPNEGO. After enabling this, you cannot access the Hadoop web consoles without a valid Kerberos ticket and correct client-side configuration.

Before you begin

- You must enable Hadoop TLS/SSL Enabled in CORE_SETTINGS before configuring the HDFS properties.
- Enabling TLS/SSL on HDFS is required before enabling TLS/SSL on YARN.

Procedure to enable Hadoop TLS/SSL Enabled

1. Log in to Cloudera Manager.
2. Navigate to Clusters.
3. Select CORE_SETTINGS.
4. Go to Configurations.
5. Search for Hadoop TLS/SSL Enabled and select the checkbox.
6. Click Save Changes.

Procedure to configure the HDFS properties

1. Log in to Cloudera Manager.
2. Navigate to Clusters.

3. Select the HDFS service
4. Go to Configurations.
5. Search for TLS/SSL.
6. Search for the following properties and configure them according to your cluster configuration:
   a. Hadoop TLS/SSL Server Keystore File Location
   b. Hadoop TLS/SSL Server Keystore File Password
   c. Hadoop TLS/SSL Server Keystore Key Password

Property	Description
Hadoop TLS/SSL Server Keystore File Location	Path to the keystore file containing the server certificate and private key.
Hadoop TLS/SSL Server Keystore File Password	Password for the server keystore file.
Hadoop TLS/SSL Server Keystore Key Password	Password that protects the private key contained in the server keystore.

7. Click Save Changes.

Procedure to configure the TLS/SSL Client TrustStore properties:

If you are not using the default trustore, then perform the below steps:

1. Log in to Cloudera Manager.
2. Navigate to Clusters.
3. Select the HDFS service
4. Go to Configurations.
5. Search for TLS/SSL.
6. Search for the following properties and configure them according to your cluster configuration:
   a. Cluster-Wide Default TLS/SSL Client Truststore Location
   b. Cluster-Wide Default TLS/SSL Client Truststore Password

Property	Description
Cluster-Wide Default TLS/SSL Client Truststore Location	Path to the client truststore file. This truststore contains certificates of trusted servers, or of Certificate Authorities trusted to identify servers.
Cluster-Wide Default TLS/SSL Client Truststore Password	Password for the client truststore file.

Important: The HDFS properties below define a cluster-wide default truststore that YARN can override.

7. Click Save Changes.

Procedure to enable web UI authentication

If you want to enable web UI authentication for the HDFS service, perform the below steps:

1. Log in to Cloudera Manager.
2. Navigate to Clusters.
3. Select the HDFS service
4. Go to Configurations.
5. Search for the Enable Authentication for HTTP Web-Consoles property.
6. Select the checkbox.

Note: This is effective only if security is enabled for the HDFS service.

7. Click Save Changes.

Configuring TLS/SSL for YARN

If you enabled TLS/SSL for HDFS, you must also enable it for YARN.

About this task

If you enable TLS/SSL for HDFS, you must also enable it for YARN.

Cloudera recommends to enable Web UI authentication for YARN.

Procedure

1. In Cloudera Manager, select the YARN service.
2. Click the Configuration tab.
3. Search for TLS/SSL.
4. Find and edit the following properties according to your cluster configuration:

Property	Description
Hadoop TLS/SSL Server Keystore File Location	Path to the keystore file containing the server certificate and private key.
Hadoop TLS/SSL Server Keystore File Password	Password for the server keystore file.
Hadoop TLS/SSL Server Keystore Key Password	Password that protects the private key contained in the server keystore.

If you want to override the cluster-wide defaults set by the HDFS properties, do the following:

5. Configure the following TLS/SSL client truststore properties for YARN.

Property	Description
TLS/SSL Client Truststore File Location	Path to the client truststore file. This truststore contains certificates of trusted servers, or of Certificate Authorities trusted to identify servers.
TLS/SSL Client Truststore File Password	Password for the client truststore file.

If you want to enable the TSL/SSL for YARN Queue Manager, do the following:

Note: If Cloudera Manager is TLS/SSL enabled, then ensure that YARN Queue Manager is also TLS/SSL enabled. Disabling Queue Manager TLS/SSL where, Cloudera Manager has TLS/SSL causes error.

6. In Cloudera Manager, select the QUEUE MANAGER service.
7. Click the Configuration tab.
8. Search for TSL/SSL.
9. Select the Enable TLS/SSL for YARN Queue Manager Store checkbox to encrypt communication between clients and YARN Queue Manager Store .
10. Configure the following properties according to your cluster configuration:

Property	Description
TLS/SSL Server JKS Keystore File Location	Path to the JKS keystore file containing the server and private key.
TLS/SSL Server JKS Keystore File Password	Password for the JKS keystore file.

11. Select the Enable TLS/SSL for YARN Queue Manager Webapp checkbox to encrypt communication between clients and YARN Queue Manager Webapp.

12. Configure the following properties according to your cluster configuration:

Property	Description
Webapp TLS/SSL Server JKS Keystore File Location	Path to the JKS keystore file containing the server and private key.
Webapp TLS/SSL Server JKS Keystore File Password	Password for the Webapp JKS keystore file.
Webapp TLS/SSL Client Trust Store File	Path to the client JKS truststore file. This truststore contains certificates of trusted servers, or of Certificate Authorities trusted to identify servers.
Webapp TLS/SSL Client Trust Store Password	Password for the Webapp truststore file.

If you want to enable Web UI authentication for YARN, do the following:

13. Search for web consoles.
14. Find the Enable Authentication for HTTP Web-Consoles property.
15. Check the property to enable web UI authentication.

Note: This is effective only if security is enabled for the HDFS service.

16. Click Save Changes.
17. Go back to the home page, by clicking the Cloudera Manager logo.
18. Select the HDFS service.
19. Click the Configuration tab.
20. Search for Hadoop SSL Enabled.
21. Find and select the Hadoop SSL Enabled property.
    The SSL communication for HDFS and YARN is enabled.
22. Click Save Changes.
23. Click the *Stale Service Restart* icon that is next to the service to invoke the cluster restart wizard.
24. Click Restart Stale Services.
25. Select Re-deploy client configuration.
26. Click Restart Now.

Configuring TLS/SSL encryption manually for Zeppelin

You can enable TLS manually for the Apache Zeppelin Server.

Before you begin

Important:

Zeppelin has been deprecated and no longer available in Cloudera Runtime 7.1.9, and is no longer supported by Cloudera.

Procedure

1. In Cloudera Manager, select the Zeppelin service from the Clusters drop-down menu.
2. In the Configuration tab, enter tls into the search box.
3. Edit the following property fields as needed for your cluster and environment:

Property	Description
Enable TLS/SSL for Zeppelin Server	Encrypt communication between clients and Zeppelin Server using Transport Layer Security (TLS) (formerly known as Secure Socket Layer (SSL)).

Property	Description
Zeppelin Server 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 Zeppelin Server is acting as a TLS/SSL server. The keystore must be in JKS format.
Zeppelin Server TLS/SSL Server JKS Keystore File Password	The password for the Zeppelin Server JKS keystore file.
Zeppelin Server TLS/SSL Trust Store File	The location on disk of the trust store, in .jks format, used to confirm the authenticity of TLS/SSL servers that Zeppelin Server 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.
Zeppelin Server TLS/SSL Trust Store Password	The password for the Zeppelin Server 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.

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

Configure ZooKeeper TLS/SSL using Cloudera Manager

TLS/SSL encryption between the ZooKeeper client and the ZooKeeper server and within the ZooKeeper Quorum is supported.

About this task

The ZooKeeper TLS/SSL feature has the following limitations:

- In each ZooKeeper server process the same ZooKeeper Server KeyStore/TrustStore configuration is used for both QuorumSSL and ClientSSL.
- HTTPS for the ZooKeeper REST Admin Server is still not supported. Even if you enable SSL for ZooKeeper, the AdminServer will still use HTTP only.

TLS/SSL encryption is automatically enabled when AutoTLS is enabled. As a result it is enabled by default in Data Hub cluster templates.

You can disable, enable and configure ZooKeeper TLS/SSL manually using Cloudera Manager:

Procedure

1. In Cloudera Manager, select the ZooKeeper service.
2. Click the Configuration tab.
3. Search for SSL.
4. Find the Enable TLS/SSL for ZooKeeper property and select it to enable TLS/SSL for ZooKeeper.

   When TLS/SSL for ZooKeeper is enabled, two ZooKeeper features get enabled:

   - QuorumSSL: the ZooKeeper servers are talking to each other using secure connection.
   - ClientSSL: the ZooKeeper clients are using secure connection when talking to the ZooKeeper server.
5. Find the Secure Client Port property and change the port number if necessary.

   When ClientSSL is enabled, a new secure port is opened on the ZooKeeper server which handles the SSL connections. Its default value is 2182.

   The old port (default: 2181) will still be available for unsecured connections. Unsecure port cannot be disabled in Cloudera Manager, because most components cannot support ZooKeeper TLS/SSL yet.
6. Click Save Changes.

7. Restart.

What to do next

Enable and configure ClientSSL by other components that support this feature.

The following components support ZooKeeper TLS/SSL:

- HBase Indexer (KS Indexer)
- Kafka
- Oozie
- Solr

Manually Configuring TLS Encryption on the Agent Listening Port

The agent listening port (TCP Port 9000) of a Cloudera Manager Agent can be secured with TLS. This port is used for retrieving diagnostic and log information.

About this task

The requirements for a Cloudera Manager Agent to enable the agent listening port are as follows:

- The following properties must be defined in the config.ini file of the Cloudera Manager Agent: use_tls=1, verify_cert_file, client_cert_file, client_keypw_file.
- An encryption key must be configured.
- A certificate must be configured.

The main requirement for the Cloudera Manager Server to connect with TLS to the agent listening port is as follows:

Procedure

- The Cloudera Manager TLS/SSL Client Trust Store File property must be configured to specify the CA certificate using which all the agent certificates are signed.

  Note: If either the Cloudera Manager Agent or the Cloudera Manager Server is not configured properly, the various diagnostic capture features in Cloudera Manager could fail.

  To verify whether the agent listening port is secured with TLS, run the following command:

  ```
  openssl s_client -connect <hostname>:9000
  ```

  If the output of this command includes a server certificate in PEM format, then the port is secured with TLS.

Enabling TLS 1.2 for database connections

Cloudera Manager supports TLS 1.2 connections to backend databases for Hadoop services like Data Explorer, Oozie, Ranger etc.

You can enhance the security of the connections created between Cloudera and backend databases such as mysql. This document describes how to enable TLS 1.2 on the databases, in Cloudera Manager, and the connection between the services and databases.

Enabling TLS 1.2 on Database server

Perform the following steps for managing the configuration of the Database Server and setting up TLS 1.2 connections to ensure the secure and proper functioning of the database environment.

Enabling TLS 1.2 on MySQL database

Refer to Enabling TLS 1.2 for MySQL Database server.

Related Information

Enabling TLS 1.2 for MySQL Database Server

Enabling TCPS on Oracle database

Refer to Enabling TCPS (TLS 1.2) for Oracle Database Server.

Related Information

Enabling TCPS for Oracle Database Server

Enabling TLS 1.2 on MariaDB

Refer to Enabling TLS 1.2 for MariaDB Database server.

Related Information

Enabling TLS 1.2 for MariaDB Database Server

Enabling TLS 1.2 on PostgreSQL

Refer to Enabling TLS 1.2 for PostgreSQL Database server.

Related Information

Enabling TLS 1.2 for PostgreSQL Database Server

Configuring TLS 1.2 for Cloudera Manager

Cloudera recommends that you secure the network connection between the Cloudera Manager server and the backend database using TLS (Transport Layer Security) 1.2 encryption. Failure to do this will result in exposing vulnerabilities to various advanced cryptographic threats. Use the following topics to learn how to enable TLS 1.2 between the Database Server and Cloudera Manager Server.

Before you begin

The first step in enabling TLS 1.2 for Cloudera Manager is to configure the Database Server to accept TLS 1.2 connections. By configuring the Database Server to accept TLS 1.2 connections, it is ready to establish secure connections with the Cloudera Manager. To enable TLS 1.2 on Database Server, perform the steps from Enabling TLS 1.2 on Database Server.

Important: Cloudera Manager does not manage the backend database, so you must perform this step before you configure TLS 1.2 in the Cloudera Manager database connection.

Enabling TLS 1.2 on Cloudera Manager Server

To enable TLS 1.2 on Cloudera Manager, perform the following procedures on Cloudera Manager Server host.

Setting up the certificate in Cloudera Manager

To set up the certificate in Cloudera Manager, refer to Setting up a certificate in Cloudera Manager.

Related Information

Setting up a certificate in Cloudera Manager

Modifying Cloudera Manager Server database configuration file

The scm_prepare_database.sh script checks the connection between the Cloudera Manager server and the database, and upon successful connection, it generates the /etc/cloudera-scm-server/db.properties file. This file includes the required configurations for the database connection.

About this task

You must follow the steps in this documenet to modify the db.properties file to add additional configuration for enabling TLS 1.2

Procedure

1. SSH into the Cloudera Manager Server host.
2. Backup the db.properties file.
3. Edit the /etc/cloudera-scm-server/db.properties file and add the following properties:

```
com.cloudera.cmf.orm.hibernate.connection.url=jdbc:mysql://localhost:3306/
mysql?useSSL=true&sslMode=VERIFY_CA &trustCertificateKeyStoreUrl=file:///
etc/cdep-ssl-conf/CA_STANDARD/truststore.jks&trustCertificateKeyStoreType
=jks&trustCertificateKeyStorePassword=cloudera&enabledTLSProtocols=TLSv1.2
```

Important:

The format of the com.cloudera.cmf.orm.hibernate.connection.url property in the /etc/cloudera-scm-server/db.properties file can vary depending on the database type. For information about the specific JDBC URL format for each database type, see JDBC URL format.

4. Restart Cloudera Manager Server.

```
sudo systemctl restart cloudera-scm-server
```

Note: If you have configured Cloudera Manager for high availability with active and passive instances of the Cloudera Manager server, repeat the above steps for the other Cloudera Manager server instance.

Configuring TLS 1.2 for Reports Manager

Cloudera recommends that you secure the network connection between the Reports Manager and the database using TLS (Transport Layer Security) 1.2 encryption. Refer to Configuring TLS 1.2 for Reports Manager to learn how to enable TLS 1.2 for Reports Manager database connections.

Related Information

Configuring TLS 1.2 for Reports Manager.

Configuring Cloudera Runtime services to connect to TLS 1.2/TCPS-enabled databases

Cloudera Manager supports TLS connection to backend databases for Hadoop Cloudera Runtime services such as Data Explorer, Ranger, Oozie, or any service that uses a backend database. Modify the service configurations in Cloudera Manager to connect the services to TLS 1.2-enabled MySQL, MariaDB, or PostgreSQL databases, or TCPS-enabled Oracle database.

Configuring Cloudera Data Explorer (Hue) to connect to TLS 1.2/TCPS-enabled databases

Learn how to configure an existing Data Explorer instance to connect to TLS-enabled MySQL, MariaDB, or PostgreSQL databases or TCPS-enabled Oracle database.

About this task

If TLS 1.2 is enabled on the database servers, and the databases are restricted or enforced to use TLS 1.2, then Data Explorer automatically uses the TLS1.2-compatible ciphers to communicate with the database securely. You do not have to configure any setting in Data Explorer's Advanced Configuration Snippet or any other configurations. This is applicable when using MySQL, MariaDB, or PostgreSQL databases as a backend database for Data Explorer.

To restrict the MySQL and MariaDB databases to use TLS 1.2, set the value of the require_secure_transport to true in the my.cnf file.

To manually enable TLS 1.2 on the Data Explorer instance, go to Cloudera Manager Clusters Hue service Configurations and select the Enable TLS/SSL for Hue option.

Attention: Ensure that you have installed the MySQL client (for MySQL or MariaDB databases) or psycopg2 Python package for the PostgreSQL database and have created database users.

The following section is specific for configuring Data Explorer to connect to a TCPS-enabled Oracle database.

Before you begin

- You must have installed and configured Oracle as a backend database for Data Explorer as described in *Using Oracle database with Hue*.
- You must have enabled TCPS on the Oracle database as described in *Enabling TCPS for Oracle Database Server*.
- You must have created database users.

Procedure

1. SSH in to the Data Explorer host as an administrator.
2. Copy the cwallet.sso file that is generated when you enabled SSL on the Oracle database to a desired location on the Data Explorer host and make sure its permissions are 644.
3. Change directory to the following:

```
cd [***ORACLE-INSTANT-CLIENT-HOME***]/network/admin
```

4. Create a file called sqlnet.ora and with the following content:

```
SSL_CLIENT_AUTHENTICATION = FALSE
WALLET_LOCATION =
            (SOURCE =
              (METHOD = FILE)
              (METHOD_DATA =
                (DIRECTORY = /[***PATH-TO-WALLET-FILE***])
              )
            )
```

5. Create a file called tnsnames.ora and with the following content:

```
ORCLPDB1_SSL =
        (DESCRIPTION =
          (ADDRESS = (PROTOCOL = TCPS)(HOST = [***HUE-DB-HOST***])(PORT =
[***HUE-DB-PORT***]))
          (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = [***SERVICE-NAME***]))
          )
          (SECURITY =
            (MY_WALLET_DIRECTORY = /[***PATH-TO-WALLET-FILE***])
          )
        )
```

6. Log in to Cloudera Manager as an Administrator.

7. Go to Clusters Hue service Configuration and add the following connection string in the Hue Database Name field:

```
(DESCRIPTION=(LOAD_BALANCE=off)(FAILOVER=on)(CONNECT_TIMEOUT=5)(TRANSPOR
T_CONNECT_TIMEOUT=3)(RETRY_COUNT=3)(ADDRESS=(PROTOCOL=TCPS)(HOST=[***HUE-
DB-HOST***])(PORT=[***HUE-DB-PORT***]))(CONNECT_DATA=(SERVICE_NAM
E=[***SERVICE-NAME***])(SECURITY =
 (MY_WALLET_DIRECTORY = /[***PATH-TO-WALLET-FILE***])))
```

   Where,

   *[***HUE-DB-HOST***]* is the FQDN of the database host

   *[***HUE-DB-PORT***]* is the port for the Data Explorer database

   *[***SERVICE-NAME***]* is the Oracle service name

   *[***PATH-TO-WALLET-FILE***]* is the location at which you have copied the wallet file (cwallet.sso) on the Data Explorer host

8. Click Save Changes.
9. Restart the Data Explorer service.

Related Information

Configuring MySQL server enforced with TLS 1.2 or higher to connect to Hue

Using Oracle database with Hue

Enabling TCPS for Oracle Database Server

Configuring Ranger to connect to TLS 1.2/TCPS-enabled databases

Updating the Ranger Database JDBC Url Override and additional configuration to connect to the secure databases.

Before you begin

Ensure that TLS 1.2 has already been enabled on the Ranger database.

Procedure

1. Go to Cloudera Manager Ranger Configuration and specify the following configuration values depending on the database type

   MySQL

Label	Configuration Name	Value
Ranger Database Type	ranger_database_type	mysql
Ranger Database User	ranger_database_user	<username>
Ranger Database User Password	ranger_database_password	<password>

Label	Configuration Name	Value
Ranger Database JDBC Url Override	ranger_database_jdbc_url	jdbc:mysql://<DB-HOST>:<DB-PORT>/<RANGER-DB-NAME>?sslMode=VERIFY_CA&trustCertificateKeyStoreU

Oracle

Label	Configuration Name	Value
Ranger Database Type	ranger_database_type	oracle
Ranger Database User	ranger_database_user	<username>
Ranger Database User Password	ranger_database_password	<password>
Ranger Database JDBC Url Override	ranger_database_jdbc_url	jdbc:oracle:thin:@tcps://<DB-HOST>:<DB-PORT>:<SERVICE_NAME>?javax.net.ssl.trustStore=<PATH_TO_TRUSTSTOR

PostgreSQL

Label	Configuration Name	Value
Ranger Database Type	ranger_database_type	postgresql
Ranger Database User	ranger_database_user	<username>
Ranger Database User Password	ranger_database_password	<password>
Ranger Database JDBC Url Override	ranger_database_jdbc_url	jdbc:postgresql://<DB-HOST>:<DB-PORT>/<RANGER-DB>?sslmode=verify-full&sslrootcert=<path-to-database-server-certificate>&enabledTLSProtocols=TLSv1.2

2. Click Save Changes.

Configuring Ranger KMS to connect to TLS 1.2/TCPS-enabled databases

Updating the Ranger KMS Database JDBC URL Override and additional configuration to connect to the secure databases.

Before you begin

Ensure that TLS 1.2 has already been enabled on the Ranger KMS database.

Procedure

1. Go to Cloudera Manager Ranger KMS Configuration and specify the following configuration values depending on the database type

   MySQL

Label	Configuration Name	Value
Ranger KMS Database Type	ranger_kms_database_type	mysql
Ranger KMS Database User	ranger_kms_database_user	<username>
Ranger KMS Database User Password	ranger_kms_database_password	<password>
Ranger KMS Database JDBC Url Override	ranger_kms_database_jdbc_url	jdbc:mysql://<DB-HOST>:<DB-PORT>/<RANGER-KMS-DB-NAME>?sslMode=VERIFY_CA&trustCertificateKeyStoreU

   Oracle

Label	Configuration Name	Value
Ranger KMS Database Type	ranger_kms_database_type	oracle
Ranger KMS Database User	ranger_kms_database_user	<username>
Ranger KMS Database User Password	ranger_kms_database_password	<password>
Ranger KMS Database JDBC Url Override	ranger_kms_database_jdbc_url	jdbc:oracle:thin:@tcps://<DB-HOST>:<DB-PORT>:<SERVICE_NAME>?javax.net.ssl.trustStore=<PATH_TO_TRUSTSTOR

   PostgreSQL

Label	Configuration Name	Value
Ranger KMS Database Type	ranger_kms_database_type	postgresql
Ranger KMS Database User	ranger_kms_database_user	<username>
Ranger KMS Database User Password	ranger_kms_database_password	<password>
Ranger KMS Database JDBC Url Override	ranger_kms_database_jdbc_url	jdbc:postgresql://<DB-HOST>:<DB-PORT>/<RANGER-KMS-DB>?sslmode=verify-full&sslrootcert=<path-to-database-server-certificate>&enabledTLSProtocols=TLSv1.2

2. Click Save Changes.

Configuring Oozie to connect to TLS 1.2/TCPS-enabled databases

You can configure Oozie's database connection properties to connect to the secure database.

Procedure

1. In Cloudera Manager Oozie Configuration , configure the database type, host, port, and database name.
2. Fine-tune your Oozie's database connection, including adding your database client trustStore or certificate location to the connection string.

   For more details , see 'Fine-tuning Oozie's database connection'.

Related Information

Fine-tuning Oozie's database connection

Configuring Streams Messaging Manager to connect to TLS 1.2/TCPS-enabled databases

Learn how you can configure an existing Streams Messaging Manager service to securely connect to its database using TLS 1.2.

Before you begin

- Ensure that TLS 1.2 has already been enabled on the Streams Messaging Manager database.
- Ensure that a truststore file containing the database certificate is available on the Streams Messaging Manager hosts. Additionally, ensure that you know the location of the file and that the user running Streams Messaging Manager has access to the file. The default user for Streams Messaging Manager is streamsmsgmgr.

Procedure

1. From Cloudera Manager, select Streams Messaging Manager service.
2. Go to Configuration Streams Messaging Manager Database JDBC Url Override and enter the following configuration values depending on the database type.

   MySQL

   ```
   jdbc:mysql://[***DB HOST***]:[***DB PORT***]/[***DB NAME***]?
   useSSL=true&trustCertificateKeyStoreUrl=file://[***TRUSTSTORE
    PATH***]&trustCertificateKeyStoreType=jks&trustCertificateKeySto
   rePassword=[***TRUSTSTORE PASSWORD***]&enabledTLSProtocols=TLS
   v1.2
   ```

   PostgreSQL

   ```
   jdbc:postgresql://[***DB HOST***]:[***DB PORT***]/[***DB
    NAME***]?useSSL=true&trustCertificateKeyStoreUrl=fil
   e://[***TRUSTSTORE PATH***]&trustCertificateKeyStoreType=jks&tr
   ustCertificateKeyStorePassword=[***TRUSTSTORE PASSWORD***]&ena
   bledTLSProtocols=TLSv1.2
   ```

   Oracle

   ```
   jdbc:oracle:thin:@tcps://[***DB HOST***]:[***DB PORT***]/[***DB
    NAME***]?javax.net.ssl.trustStore=[***TRUSTSTORE PATH***]&jav
   ax.net.ssl.trustStorePassword=[***TRUSTSTORE PASSWORD***]&oracle.
   net.ssl_server_dn_match=false
   ```

   - Replace *[***DB HOST***]*, *[***DB PORT***]*, and *[***DB NAME***]* with the host, port, and name of the database.

- Replace *[***TRUSTSTORE PATH***]* with the full path to a truststore that contains the database certificate. The truststore must be available on the host that SMM is deployed on. Additionally, the user that the Streams Messaging Manager service runs as, default is streamsmsgmgr, must have access to the file.
- Replace *[***TRUSTSTORE PASSWORD***]* with the password used to access the truststore you specify in *[***TRUSTSTORE PATH***]*.

3. Click Save Changes.
4. Restart the Streams Messaging Manager service.

Results

The Streams Messaging Manager service establishes a secure connection with its database.

Configuring Schema Registry to connect to TLS 1.2/TCPS-enabled databases

Learn how you can configure an existing Schema Registry service to securely connect to its database using TLS 1.2.

Before you begin

- Ensure that TLS 1.2 has already been enabled on the Schema Registry database.
- Ensure that a truststore file containing the database certificate is available on the Schema Registry hosts. Additionally, ensure that you know the location of the file and that the user running Schema Registry has access to the file. The default user for Schema Registry is schemaregistry.

Procedure

1. Go to Cloudera Manager Schema Registry Configuration Schema Registry Database JDBC Url Override and enter the following configuration values depending on the database type.

   MySQL

   ```
   jdbc:mysql://[***DB HOST***]:[***DB PORT***]/[***DB NAME***]?
   useSSL=true&trustCertificateKeyStoreUrl=file://[***TRUSTSTORE
    PATH***]&trustCertificateKeyStoreType=jks&trustCertificateKeySto
   rePassword=[***TRUSTSTORE PASSWORD***]&enabledTLSProtocols=TLS
   v1.2
   ```

   PostgreSQL

   ```
   jdbc:postgresql://[***DB HOST***]:[***DB PORT***]/[***DB
    NAME***]?useSSL=true&trustCertificateKeyStoreUrl=fil
   e://[***TRUSTSTORE PATH***]&trustCertificateKeyStoreType=jks&tr
   ustCertificateKeyStorePassword=[***TRUSTSTORE PASSWORD***]&ena
   bledTLSProtocols=TLSv1.2
   ```

   Oracle

   ```
   jdbc:oracle:thin:@tcps://[***DB HOST***]:[***DB PORT***]/[***DB
    NAME***]?javax.net.ssl.trustStore=[***TRUSTSTORE PATH***]&jav
   ax.net.ssl.trustStorePassword=[***TRUSTSTORE PASSWORD***]&oracle.
   net.ssl_server_dn_match=false
   ```

   - Replace *[***DB HOST***]*, *[***DB PORT***]*, and *[***DB NAME***]* with the host, port, and name of the database.
   - Replace *[***TRUSTSTORE PATH***]* with the full path to a truststore that contains the database certificate. The truststore must be available on the host that Schema Registry is deployed on. Additionally, the user that the Schema Registry service runs as, default is schemaregistry, must have access to the file.
   - Replace *[***TRUSTSTORE PASSWORD***]* with the password used to access the truststore you specify in *[***TRUSTSTORE PATH***]*.

2. Click Save Changes.

3. Restart the SMM service.

Results

The Schema Registry service establishes a secure connection with its database.

Configuring Hive to connect to TLS 1.2/TCPS-enabled databases

If you have enabled TLS on your database, then learn how to update the Hive Database JDBC URL Override property to connect to the secure databases.

Procedure

1. Go to Cloudera Manager Hive Configuration Hive Metastore Server Advanced Configuration Snippet (Safety Valve) for hive-site.xml .
2. Configure the javax.jdo.option.ConnectionURL property with values depending on the database type.
   - MySQL

     ```
     jdbc:mysql://[***DB-HOST***]:[***DB-PORT***]/[***DB-NAME***]?sslMode=VER
     IFY_CA&trustCertificateKeyStoreUrl=file://[***TRUSTSTORE-PATH***]&trustC
     ertificateKeyStoreType=jks&trustCertificateKeyStorePassword=[***TRUSTSTO
     RE-PASSWORD***]&enabledTLSProtocols=TLSv1.2
     ```

   - PostgreSQL

     ```
     jdbc:postgresql://[***DB-HOST***]:[***DB-PORT***]/[***DB-NAME***]?sslMod
     e=VERIFY_CA&trustCertificateKeyStoreUrl=file://[***TRUSTSTORE-PATH***]&t
     rustCertificateKeyStoreType=jks&trustCertificateKeyStorePassword=[***TRU
     STSTORE-PASSWORD***]&enabledTLSProtocols=TLSv1.2
     ```

   - MariaDB

     ```
     jdbc:mysql://[***DB-HOST***]:[***DB-PORT***]/[***DB-NAME***]?sslMode=VER
     IFY_CA&trustCertificateKeyStoreUrl=file://[***TRUSTSTORE-PATH***]&trustC
     ertificateKeyStoreType=jks&trustCertificateKeyStorePassword=[***TRUSTSTO
     RE-PASSWORD***]&enabledTLSProtocols=TLSv1.2
     ```

   - Oracle TCPS

     ```
     jdbc:oracle:thin:@tcps://[***DB-HOST***]:[***DB-PORT***]/[***DB-NAME***]
     ?javax.net.ssl.trustStore=[***TRUSTSTORE-PATH***]&javax.net.ssl.trustSto
     rePassword=[***TRUSTSTORE-PASSWORD***]&oracle.net.ssl_server_dn_match=fa
     lse
     ```

   Where,

   - [***DB-HOST***], [***DB-PORT***], and [***DB-NAME***] represent the Host, Port, and Database name used for the Hive Metastore service.
   - [***TRUSTSTORE-PATH***] represents the path to the Java truststore file.
   - [***TRUSTSTORE-PASSWORD***] represents the password used to access the Java truststore file.
3. Click Save Changes.

How to connect Cloudera components to a TCPS-enabled Oracle database

Learn which Cloudera components support TCPS-enabled Oracle database and how to configure them to work with an TCPS-enabled Oracle database.

What is TCPS

TCPS is Transmission Control Protocol with SSL. It provides higher security between the database and Cloudera services than TCP alone.

List of Cloudera components and Cloudera Runtime services that support TCPS-enabled Oracle database

The following Cloudera components can use a TCPS-enabled Oracle database starting with CDP 7.1.9:

- Cloudera Manager server
- Reports Manager
- Hive MetaStore
- Cloudera Data Explorer (Hue)
- Schema Registry
- Streams Messaging Manager
- Oozie
- Sqoop
- Ranger
- Ranger KMS

High-level steps to configure and set up TCPS

In any TLS connection, there are two entities involved–a client and a server. In Cloudera, Cloudera Manager and Cloudera Runtime services are the clients and the Oracle database is the server.

1. First, you enable TCPS on the Oracle database server, as described in Enabling TCPS for Oracle Database Server.
2. Specify the JDBC URL or a connection string when you add the Cloudera Runtime services in the **Add service** wizard using Cloudera Manager. Review the instructions for each supported service in Configuring runtime services to connect to TLS 1.2/TCPS-enabled databases.

You can also configure the existing Cloudera Runtime services to connect to a TCPS-enabled Oracle database. See Configuring runtime services to connect to TLS 1.2/TCPS-enabled databases.

How to verify whether TCPS is enabled on your database

You can verify whether TCPS is successfully enabled on your Oracle database by running the following command:

```
SQL> select sys_context('userenv','network_protocol') from dual;
```

If TCPS is enabled, you see the following output:

```
SYS_CONTEXT('USERENV','NETWORK_PROTOCOL')
------------------------------------------------------------------------
tcps
```

Alternatively, check whether the PROTOCOL = TCPS line is present in the following configuration files:

- listener.ora
- tnsnames.ora