## 

## **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**

| 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. See [Generate TLS Certificates](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_gkg_xs3_lx) for details. |
| Internal CA-signed certificates | This type of certificate is signed by your organization's internal CA. Organizations using [OpenSSL Certificate Authority](https://jamielinux.com/docs/openssl-certificate-authority/), Microsoft [Active Directory Certificate Service](https://technet.microsoft.com/en-us/windowsserver/dd448615.aspx), or another internal CA system can use this type of certificate. See [How to Configure TLS Encryption for Cloudera Manager](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_wk4_jlx_qw) for details about using internal CA-signed certificates for configuration. |
| 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. See [Using Self-signed Certificates for TLS](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/sg_self_signed_tls.html#sg_self_signed_tls) for details. |
|  |  |
|  |  |

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:

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)](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/cm_sg_create_key_trust.html#certificate_formats) supported by the host. For information about converting formats, see [How to Convert Certificate Encodings (DER, JKS, PEM) for TLS/SSL](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/cm_sg_openssl_jks.html#xd_583c10bfdbd326ba-7dae4aa6-147c30d0933--7a5f)

## **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.

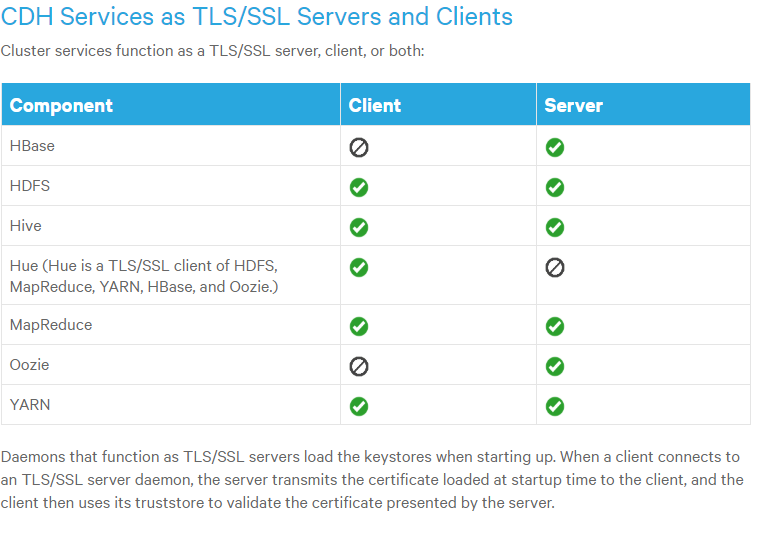
The standard Oracle Java JDK distribution includes a default **truststore** (cacerts) that contains root certificates for many well-known CAs, including Symantec. Rather than using the default truststore, Cloudera recommends using the **alternative truststore**, jssecacerts. The alternative truststore 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 [Generate TLS Certificates](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_gkg_xs3_lx) for details.

The private keys are maintained in the **keystore**.

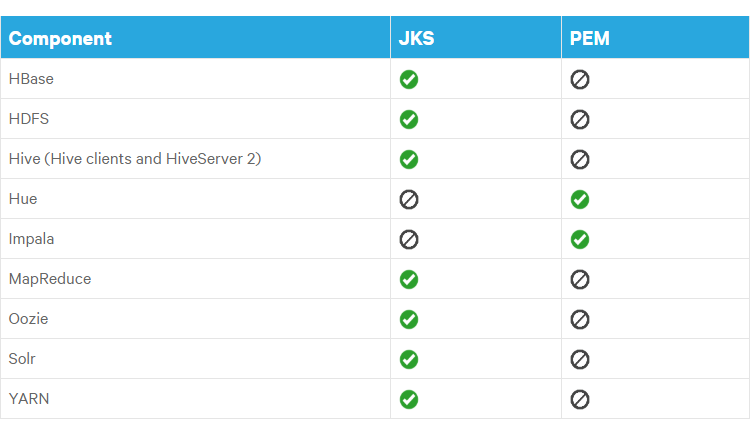
Although the keystore and truststore in some environments may comprise the same file, as configured for Cloudera Manager Server and CDH clusters, the keystore and truststore are distinct files. For Cloudera Manager Server clusters, each host should have its own keystore, while several hosts can share the same truststore. 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 jssecacerts 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. |



## **Certificate Formats (JKS, PEM) and Cluster Components**

Cloudera Manager Server, Cloudera Management Service, and many other CDH services use JKS formatted keystores and certificates. Cloudera Manager Agent, Hue, Key Trustee Server, Impala, and other Python or C++ based services require [PEM](https://en.wikipedia.org/wiki/Privacy-enhanced_Electronic_Mail) formatted certificates and keystores rather than Java.



## **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.

**TLS levels:**

Transport Layer Security (TLS) provides encryption and authentication in the communications between the Cloudera Manager Server and Agents. Encryption prevents snooping of communications, and authentication helps prevent malicious servers or agents from causing problems in your cluster.

Cloudera Manager supports three levels of TLS security. It is necessary to work through the configuration of Level 1, and then Level 2 TLS to be able to configure Level 3 encryption. The configurations build on each other to reach Level 3 which is the strongest level of TLS security.

* Level 1 (Good) - This level only configures encrypted communication between the browser and Cloudera Manager, and between Agents and the Cloudera Manager Server. See [Configuring TLS Encryption Only for Cloudera Manager](https://docs.cloudera.com/documentation/enterprise/5-2-x/topics/cm_sg_tls_browser.html#xd_583c10bfdbd326ba-7dae4aa6-147c30d0933--7a61) followed by [Level 1: Configuring TLS Encryption for Cloudera Manager Agents](https://docs.cloudera.com/documentation/enterprise/5-2-x/topics/cm_sg_config_tls_encr.html#topic_2) for instructions. Level 1 encryption prevents snooping of commands and controls ongoing communication between the Agents and Cloudera Manager.
* Level 2 (Better) - This level includes encrypted communication between the Agents and the Server, as well as strong verification of the Cloudera Manager Server certificate by the Agents. See [Level 2: Configuring TLS Verification of Cloudera Manager Server by the Agents](https://docs.cloudera.com/documentation/enterprise/5-2-x/topics/cm_sg_config_tls_auth.html#topic_3). Level 2 provides Agents with an additional level of security by verifying trust for the certificate presented by the Cloudera Manager Server.
* Level 3 (Best) - Encrypted communication between the Agents and the Server. Level 3 TLS includes encrypted communication between the Agents and the Server, strong verification of the Cloudera Manager Server certificate by the Agents and authentication of Agents to the Cloudera Manager Server using self-signed or CA-signed certs. See [Level 3: Configuring TLS Authentication of Agents to the Cloudera Manager Server](https://docs.cloudera.com/documentation/enterprise/5-2-x/topics/cm_sg_config_tls_agent_auth.html#topic_4). Level 3 addresses the untrusted network scenario where you need to prevent cluster Servers being spoofed by untrusted Agents running on a host. Cloudera recommends you configure Level 3 TLS encryption for untrusted network environments before enabling Kerberos authentication. This provides secure communication of keytabs between the Cloudera Manager Server and verified Agents across the cluster.

**Important:**

* Cloudera strongly recommends that you set up a fully-functional CDH cluster and Cloudera Manager before you begin configuring the Cloudera Manager Server and Agents to use TLS.
* Cloudera Manager will continue to accept HTTP requests on port 7180 (default) but will immediately redirect clients to port 7183 for HTTPS connectivity once TLS is enabled.
* Once Level 3 TLS is configured, if you want to add new hosts running Agents, you must manually deploy the Cloudera Manager agent and daemon's packages for your platform, issue a new certificate for the host, configure /etc/cloudera-scm-agent/config.ini to use SSL/TLS and then bring the host online.

Conversely, you can disable TLS to add the host, configure the new host for TLS, then re-enable with the proper configuration in place. Either approach is valid, based on your needs.

* For all hosts running Agents, Cloudera recommends you start with creating the keystore in Java first, and then exporting the key and certificate using openSSL for use by the Agent or Hue.

## **Configure TLS for the Cloudera Manager Admin Console**

Use the following procedure to enable TLS encryption for the Cloudera Manager Server admin interface.

* [Step 1: Enable HTTPS for the Cloudera Manager Admin Console](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_gj4_g5d_xn)
* [Step 2: Specify SSL Truststore Properties for Cloudera Management Services](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_pzt_g1q_5q)
* [Step 3: Restart Cloudera Manager and Services](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_xkr_l1q_5q)

### Step 1: Enable HTTPS for the Cloudera Manager Admin Console

1. Log in to the Cloudera Manager Admin Console.
2. Select **Administration** > **Settings**.
3. Select the **Security** category.

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

1. Configure the following TLS settings:
2. Enter a **Reason for Change**, then click **Save Changes** to save the settings.

### Step 2: Specify SSL Truststore Properties for Cloudera Management Services

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.

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:  **/opt/hadoop/security/jks/truststore.jks** |
| Cloudera Manager Server TLS/SSL Certificate Trust Store Password | The password for the truststore file. |

1. Click **Save Changes** to commit the changes.

### Step 3: Restart Cloudera Manager and Services

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.

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

1. 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.
2. Restart the Cloudera Management Service (**Cloudera Management Service** > **Actions** > **Restart**).

## **Configure TLS for Cloudera Manager Agents(level-1)**

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](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_y3m_l4q_3p)
* [Step 2: Enable TLS on Cloudera Manager Agent Hosts](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_a3w_g5d_xn)
* [Step 3: Restart Cloudera Manager Server and Agents](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_qx3_j5d_xn)
* [Step 4: Verify that the Cloudera Manager Server and Agents are Communicating](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#id_hrm_t5d_xn)

### Step 1: Enable TLS Encryption for Agents in Cloudera Manager

Configure the TLS properties for Cloudera Manager Agents.

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. Click **Save Changes** to commit the changes.

### Step 2: Enable TLS on Cloudera Manager Agent Hosts

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.

1. 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

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

1. 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

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.

NOTE: if heartbeat is not receiving ,then check config.ini.For admin server full hostname should be there.Because we have generated our certificates including fqdn.

## **Enable Server Certificate Verification on Cloudera Manager Agents(level-2)**

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:

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:

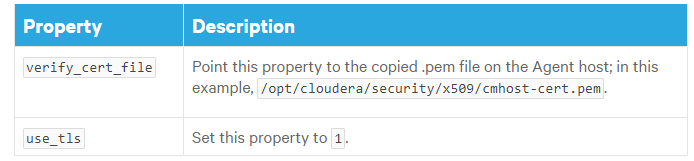
verify\_cert\_file= /opt/hadoop/security/ca-certs/rootCA.pem

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.

Or

Agents can verify the Cloudera Manager Server using either the Server certificate or the associated root CA certificate. Do one of the following to proceed:

* **Copy the Cloudera Manager Server .pem file to the Agent host**
  1. For verification by the Agent, copy the Server .pem file (for example, cmhost.pem ) to any directory on the Agent host. In the examples, this path is /opt/cloudera/security/x509/cmhost.pem.
  2. On the Agent host, open the /etc/cloudera-scm-agent/config.ini configuration file and edit the following properties.



1. 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

1. 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(level-3)**

**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](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_ek3_sdl_rp)
* [Step 2: Create a Password File](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_dsv_q3w_wn)
* [Step 3: Configure the Agent to Use Private Keys and Certificates](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_ym5_q3w_wn)
* [Step 4: Enable Agent Certificate Authentication](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_xtp_q3w_wn)
* [Step 5: Restart Cloudera Manager Server and Agents](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_anx_ypw_wn)
* [Step 6: Verify that Cloudera Manager Server and Agents are Communicating](https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/how_to_configure_cm_tls.html#concept_htk_q3w_wn)

### Step 1: Export the Private Key to a File

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:

1. Export the private key and certificate:

sudo /usr/java/jdk1.8.0\_251-amd64/bin/keytool \

-importkeystore \

-srckeystore /opt/hadoop/security/jks/$i-keystore.jks \

-srcstorepass password \

-srckeypass password \

-destkeystore /opt/hadoop/security/tmp/$i-keystore.p12 \

-deststoretype PKCS12 \

-srcalias $i \

-deststorepass password \

-destkeypass password

done

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

sudo openssl pkcs12 -in /opt/hadoop/security/tmp/$i-keystore.p12 \

-passin pass:password \

-passout pass:password \

-nocerts \

-out /opt/hadoop/security/x509/$i.key

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

sudo ln -s /opt/hadoop/security/x509/`hostname -f`.key /opt/hadoop/security/x509/key.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.

**\*we have already completed this step in our TLS/SSL text file.Do not do this again.**

### Step 2: Create a Password File

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:

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

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

1. 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

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/hadoop/security/x509/key.pem | 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/hadoop/security/x509/cert.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

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** | **/opt/hadoop/security/jks/truststore.jks** |
| **Cloudera Manager TLS/SSL Certificate Trust Store Password** | Specify the password for the  truststore. |

1. Click **Save Changes** to commit the changes.

### Step 5: Restart Cloudera Manager Server and Agents

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

1. 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

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    Heartbeating 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.

**TLS/SSL for cloudera services:**

**Enabling SSL For HDFS And Yarn:**

in CM do the following:

HDFS -🡪config

1. **hadoop.rpc.protection** set to **PRIVACY**

2. **dfs.encrypt.data.transfer** 🡪 **enable**

3. **dfs.data.transfer.protection** is **\*unset\*** (it is superceded by dfs.data.transfer.protection)

Additionally, enable SSL:

Check **hadoop.ssl.enabled**

set **ssl.server.keystore.location** to **/opt/hadoop/security/jks/keystore.jks**

keystore passwords are password:

**ssl.server.keystore.password**

**ssl.server.keystore.keypassword**

**ssl.client.truststore.location** is **/opt/hadoop/security/jks/truststore.jks**

password is password

**YARN:**

set keystore file **ssl.server.keystore.location** to **/opt/hadoop/security/jks/keystore.jks**

set keystore password **ssl.server.keystore.password** and **ssl.server.keystore.keypassword**

deploy client configuration and start ONLY HDFS, Zookeeper and YARN.

As a basic test to see if this worked, open webconsole of RM,it its direct to https then ssl is enabled successfully.

Other method is to submit a sample mapreduce job.If job fails then ssl is not configured properly.

**Submit mapred job:**

[Devendra@hadoop-slave-1 ~]$ hadoop jar /opt/cloudera/parcels/CDH/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar pi 10 10

Number of Maps = 10

Samples per Map = 10

Wrote input for Map #0

Wrote input for Map #1

Wrote input for Map #2

Wrote input for Map #3

Wrote input for Map #4

Wrote input for Map #5

Wrote input for Map #6

Wrote input for Map #7

Wrote input for Map #8

Wrote input for Map #9

Starting Job

20/06/20 14:45:44 INFO client.RMProxy: Connecting to ResourceManager at hadoop-slave-1.us-central1-a.c.hadoop-278014.internal/10.128.0.3:8032

20/06/20 14:45:45 INFO hdfs.DFSClient: Created token for Devendra: HDFS\_DELEGATION\_TOKEN owner=Devendra@HADOOPSECURITY.COM, renewer=yarn, realUser=, issueDate=1592664345154, maxDate=1593269145154, sequenceNumber=1, masterKeyId=20 on 10.128.0.3:8020

20/06/20 14:45:45 INFO security.TokenCache: Got dt for hdfs://hadoop-slave-1.us-central1-a.c.hadoop-278014.internal:8020; Kind: HDFS\_DELEGATION\_TOKEN, Service: 10.128.0.3:8020, Ident: (token for Devendra: HDFS\_DELEGATION\_TOKEN owner=Devendra@HADOOPSECURITY.COM, renewer=yarn, realUser=, issueDate=1592664345154, maxDate=1593269145154, sequenceNumber=1, masterKeyId=20)

20/06/20 14:45:45 INFO input.FileInputFormat: Total input paths to process : 10

20/06/20 14:45:45 INFO mapreduce.JobSubmitter: number of splits:10

20/06/20 14:45:46 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_1592664187820\_0001

20/06/20 14:45:46 INFO mapreduce.JobSubmitter: Kind: HDFS\_DELEGATION\_TOKEN, Service: 10.128.0.3:8020, Ident: (token for Devendra: HDFS\_DELEGATION\_TOKEN owner=Devendra@HADOOPSECURITY.COM, renewer=yarn, realUser=, issueDate=1592664345154, maxDate=1593269145154, sequenceNumber=1, masterKeyId=20)

20/06/20 14:45:48 INFO impl.YarnClientImpl: Submitted application application\_1592664187820\_0001

20/06/20 14:45:48 INFO mapreduce.Job: The url to track the job: https://hadoop-slave-1.us-central1-a.c.hadoop-278014.internal:8090/proxy/application\_1592664187820\_000

Job Finished in 51.828 seconds

Estimated value of Pi is 3.20000000000000000000

Our job completed without error.So ssl is configured properly.

**SSL/SASL on hive:**

This topic describes how to set up encrypted communication between HiveServer2 and its clients. Encrypting Hive communication depends on whether you are using Kerberos authentication for communications between HiveServer2 and JDBC/ODBC client drivers.

## **With Kerberos Enabled**

With Kerberos authentication enabled, traffic between the Hive JDBC or ODBC drivers and HiveServer2 can be encrypted using SASL-QOP which allows you to preserve both data integrity (using checksums to validate message integrity) and confidentiality (by encrypting messages)

## **Without Kerberos Enabled**

If you are using any alternate means of authentication, such as [LDAP](https://docs.cloudera.com/documentation/enterprise/5-3-x/topics/cdh_sg_hiveserver2_security.html#topic_9_1_3), between HiveServer2 and its clients, you can configure Secure Socket Layer (SSL) communication between them. For instructions, see [Configuring Encrypted Client/Server Communication for non-Kerberos HiveServer2 Connections](https://docs.cloudera.com/documentation/enterprise/5-3-x/topics/sg_hive_encryption.html#concept_rqh_sff_cm).

## **Configuring Encrypted Client/Server Communication for Kerberos-enabled HiveServer2 Connections**

With Kerberos authentication enabled, traffic between the Hive JDBC or ODBC drivers and HiveServer2 can be encrypted which allows you to preserve data integrity (using checksums to validate message integrity) and confidentiality (by encrypting messages). This can be enabled by setting the hive.server2.thrift.sasl.qop property in hive-site.xml. For example,

<property>

<name>hive.server2.thrift.sasl.qop</name>

<value>auth-conf</value>

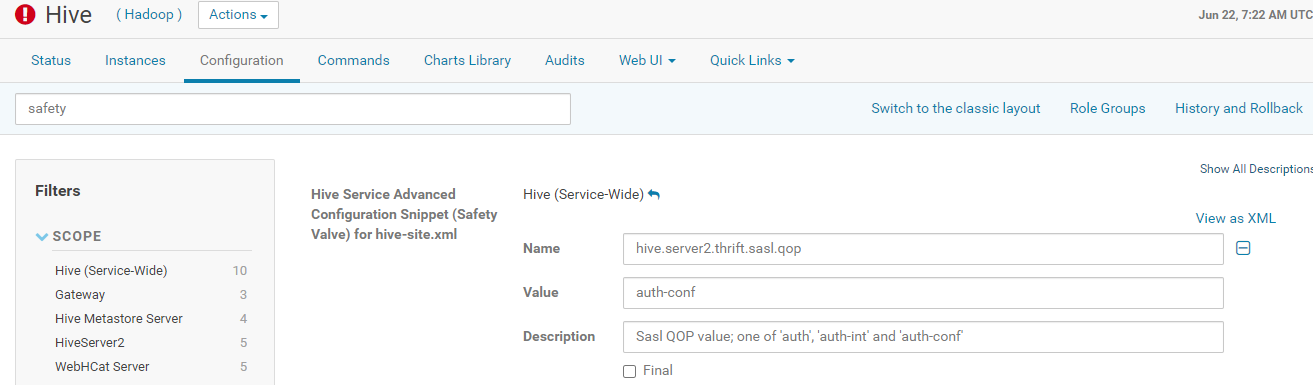
<description>Sasl QOP value; one of 'auth', 'auth-int' and 'auth-conf'</description>

</property>

Valid settings for the value field are:

* auth: Authentication only (default)
* auth-int: Authentication with integrity protection
* auth-conf: Authentication with confidentiality protection

go to hive🡪 config 🡪 add above value in safety valve for **Hive Service Advanced Configuration Snippet**



**Validate SASL via beeline:**

The parameter value that you specify above in the HiveServer2 configuration, should match that specified in the Beeline client connection JDBC URL,otherwise encrypted session will not start. For example:

!connect jdbc:hive2://hadoop-slave-1.us-central1-a.c.hadoop-278014.internal:10000/retail;principal=hive/hadoop-slave-1.us-central1-a.c.hadoop-278014.internal@HADOOPSECURITY.COM;**saslQop=auth-conf**

beeline> !connect jdbc:hive2://hadoop-slave-1.us-central1-a.c.hadoop-278014.internal:10000/retail;principal=hive/hadoop-slave-1.us-central1-a.c.hadoop-278014.internal@HADOOPSECURITY.COM;saslQop=auth-conf

scan complete in 3ms

Connecting to jdbc:hive2://hadoop-slave-1.us-central1-a.c.hadoop-278014.internal:10000/retail;principal=hive/hadoop-slave-1.us-central1-a.c.hadoop-278014.internal@HADOOPSECURITY.COM;saslQop=auth-conf

Connected to: Apache Hive (version 1.1.0-cdh5.14.0)

Driver: Hive JDBC (version 1.1.0-cdh5.14.0)

Transaction isolation: TRANSACTION\_REPEATABLE\_READ

0: jdbc:hive2://hadoop-slave-1.us-central1-a.>

0: jdbc:hive2://hadoop-slave-1.us-central1-a.> show tables;

INFO : Compiling command(queryId=hive\_20200622071010\_eade2852-78ba-4888-bd7d-7b58182c0a22): show tables

INFO : Semantic Analysis Completed

INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:tab\_name, type:string, comment:from deserializer)], properties:null)

INFO : Completed compiling command(queryId=hive\_20200622071010\_eade2852-78ba-4888-bd7d-7b58182c0a22); Time taken: 0.088 seconds

INFO : Executing command(queryId=hive\_20200622071010\_eade2852-78ba-4888-bd7d-7b58182c0a22): show tables

INFO : Starting task [Stage-0:DDL] in serial mode

INFO : Completed executing command(queryId=hive\_20200622071010\_eade2852-78ba-4888-bd7d-7b58182c0a22); Time taken: 0.303 seconds

INFO : OK

+-----------+--+

| tab\_name |

+-----------+--+

| orders |

**Submit a sample job:**

0: jdbc:hive2://hadoop-slave-1.us-central1-a.> select count(\*) from orders;

INFO : Compiling command(queryId=hive\_20200622071010\_2a963436-3ffa-4378-9da4-e5fde577dff1): select count(\*) from orders

INFO : Semantic Analysis Completed

INFO : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:\_c0, type:bigint, comment:null)], properties:null)

INFO : Completed compiling command(queryId=hive\_20200622071010\_2a963436-3ffa-4378-9da4-e5fde577dff1); Time taken: 0.271 seconds

INFO : Executing command(queryId=hive\_20200622071010\_2a963436-3ffa-4378-9da4-e5fde577dff1): select count(\*) from orders

INFO : Query ID = hive\_20200622071010\_2a963436-3ffa-4378-9da4-e5fde577dff1

INFO : Total jobs = 1

INFO : Launching Job 1 out of 1

INFO : Starting task [Stage-1:MAPRED] in serial mode

INFO : Number of reduce tasks determined at compile time: 1

INFO : In order to change the average load for a reducer (in bytes):

INFO : set hive.exec.reducers.bytes.per.reducer=<number>

INFO : In order to limit the maximum number of reducers:

INFO : set hive.exec.reducers.max=<number>

INFO : In order to set a constant number of reducers:

INFO : set mapreduce.job.reduces=<number>

INFO : number of splits:1

INFO : Submitting tokens for job: job\_1592808620949\_0005

INFO : Kind: HDFS\_DELEGATION\_TOKEN, Service: 10.128.0.3:8020, Ident: (token for hive: HDFS\_DELEGATION\_TOKEN owner=hive/hadoop-slave-1.us-central1-a.c.hadoop-278014.internal@HADOOPSECURITY.COM, renewer=yarn, realUser=, issueDate=1592809856497, maxDate=1593414656497, sequenceNumber=9, masterKeyId=32)

INFO : The url to track the job: https://hadoop-slave-1.us-central1-a.c.hadoop-278014.internal:8090/proxy/application\_1592808620949\_0005/

INFO : Starting Job = job\_1592808620949\_0005, Tracking URL = https://hadoop-slave-1.us-central1-a.c.hadoop-278014.internal:8090/proxy/application\_1592808620949\_0005/

INFO : Kill Command = /opt/cloudera/parcels/CDH-5.14.0-1.cdh5.14.0.p0.24/lib/hadoop/bin/hadoop job -kill job\_1592808620949\_0005

INFO : Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1

INFO : 2020-06-22 07:11:06,798 Stage-1 map = 0%, reduce = 0%

INFO : 2020-06-22 07:11:17,903 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.89 sec

INFO : 2020-06-22 07:11:25,301 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 4.18 sec

INFO : MapReduce Total cumulative CPU time: 4 seconds 180 msec

INFO : Ended Job = job\_1592808620949\_0005

INFO : MapReduce Jobs Launched:

INFO : Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 4.18 sec HDFS Read: 3007805 HDFS Write: 6 SUCCESS

INFO : Total MapReduce CPU Time Spent: 4 seconds 180 msec

INFO : Completed executing command(queryId=hive\_20200622071010\_2a963436-3ffa-4378-9da4-e5fde577dff1); Time taken: 30.272 seconds

INFO : OK

+--------+--+

| \_c0 |

+--------+--+

| 68883 |

+--------+--+

1 row selected (30.708 seconds)

Since our job submitted successfully,means SASL is working fine.

## **Configuring Encrypted Client/Server Communication for non-Kerberos HiveServer2 Connections**

You can use either Cloudera Manager or the command-line to enable SSL encryption for non-Kerberized client connections to HiveServer2.

* [Using Cloudera Manager](https://docs.cloudera.com/documentation/enterprise/5-3-x/topics/sg_hive_encryption.html#concept_tp1_whc_dr)
* [Using the Command Line](https://docs.cloudera.com/documentation/enterprise/5-3-x/topics/sg_hive_encryption.html#concept_dzr_whc_dr)

### Using Cloudera Manager

The steps for configuring and enabling SSL for Hive are as follows:

1. Open the Cloudera Manager Admin Console and navigate to the Hive service.
2. Click **Configuration**.
3. In the Search field, type **SSL** to show the Hive SSL properties (found under the **Service-Wide** > **Security** category).
4. Edit the following SSL properties according to your cluster configuration.
5. Click **Save Changes**.
6. Restart the Hive service.

| ***Hive SSL Properties*** | |
| --- | --- |
| **Property** | **Description** |
| **Enable SSL for HiveServer** | Enable support for encrypted client-server communication using Secure Socket Layer (SSL) for HiveServer2 connections. Not applicable for Kerberos-enabled connections. |
| **Keystore File Path** | Path to the SSL keystore. |
| **Keystore Password** | Password for the keystore. |

### Using the Command Line

* To enable SSL, add the following configuration parameters to hive-site.xml :
* <property>
* <name>hive.server2.use.SSL</name>
* <value>true</value>
* <description>enable/disable SSL </description>
* </property>
* <property>
* <name>hive.server2.keystore.path</name>
* <value>keystore-file-path</value>
* <description>path to keystore file</description>
* </property>
* <property>
* <name>hive.server2.keystore.password</name>
* <value>keystore-file-password</value>
* <description>keystore password</description>

</property>

* The keystore must contain the server's certificate.
* The JDBC client must add the following properties in the connection URL when connecting to a HiveServer2 using SSL:

;ssl=true[;sslTrustStore=<Trust-Store-Path>;trustStorePassword=<Trust-Store-password>]

* Make sure one of the following is true:
  + Either: sslTrustStore points to the trust store file containing the server's certificate; for example:
  + jdbc:hive2://localhost:10000/default;ssl=true;\
  + sslTrustStore=/home/usr1/ssl/trust\_store.jks;trustStorePassword=xyz
  + or: the Trust Store arguments are set using the Java system properties javax.net.ssl.trustStore and javax.net.ssl.trustStorePassword; for example:
  + java -Djavax.net.ssl.trustStore=/home/usr1/ssl/trust\_store.jks -Djavax.net.ssl.trustStorePassword=xyz \

MyClass jdbc:hive2://localhost:10000/default;ssl=true

**TLS/SSL with HBase and oozie:**

**HBase:**

Check **hadoop.ssl.enabled**

set **ssl.server.keystore.location** to **/opt/hadoop/security/jks/keystore.jks**

keystore passwords are password:

ssl.server.keystore.password

ssl.server.keystore.keypassword

**oozie:**

**search for ssl in config**

check **Enable TLS/SSL for Oozie**

**keystore:**

set **ssl.server.keystore.location** to **/opt/hadoop/security/jks/keystore.jks**

keystore passwords are password:

ssl.server.keystore.password

ssl.server.keystore.keypassword

**trustore:**

set **ssl.server.truststore.location** to **/opt/hadoop/security/jks/truststore.jks**

truststore store passwords are password:

ssl.server.truststore.password

ssl.server.truststore.keypassword

**TLS/SSL with Impala:**

since impala is a non java program and written in c,so we will be adding x509 cert file and keynopw.pem.

**client\_services\_ssl\_enabled:** true

ssl\_server\_**certificate**: /opt/hadoop/security/x509/cert.pem

**ssl\_private\_key**: /opt/hadoop/security/x509/keynopw.pem

**TLS/SSL with hue:**

Since hue is written in python so it requires key in x509 format and it requires a different truststore.

Check **Enable TLS/SSL for Hue**

**ssl\_server\_certificate**: /opt/hadoop/security/x509/cert.pem

**ssl\_private\_key**: /opt/hadoop/security/x509/keynopw.pem

Hue Service Environment Advanced Configuration Snippet

**REQUESTS\_CA\_BUNDLE**=/opt/hadoop/security/truststore/ca-truststore.pem

Safety valve for Impala and hive added to

Hue Service Advanced Configuration Snippet (Safety Valve) for hue\_safety\_valve.ini:

[hadoop]

[[hdfs\_clusters]]

[[[default]]]

# Use WebHdfs/HttpFs as the communication mechanism.

# Domain should be the NameNode or HttpFs host.

webhdfs\_url=https://INTERNAL-HOSTNAME-TO-NAMENODE-OR-HTTPFS-SERVER:50470/webhdfs/v1

**[impala]**

[[ssl]]

# SSL communication enabled for this server.

enabled=true

# Path to Certificate Authority certificates.

cacerts=/opt/hadoop/security/truststore/ca-truststore.pem

**[beeswax]**

[[ssl]]

# SSL communication enabled for this server.

enabled=true

# Path to Certificate Authority certificates.

cacerts=/opt/hadoop/security/truststore/ca-truststore.pem

**Kafka security --🡪Enabling Kerberos and TLS for kafka:**

**Step 1: Generating Keys and Certificates for Kafka Brokers**

Generate the key and the certificate for each machine in the cluster using the Java keytool utility. See [Generate TLS Certificates](https://docs.cloudera.com/documentation/enterprise/latest/topics/how_to_configure_cm_tls.html#concept_gkg_xs3_lx).

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

**Step 2: Creating Your Own Certificate Authority**

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

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

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

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

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

keytool -keystore {client.truststore.jks} -alias CARoot -import -file {ca-cert}

### Step 3: Signing the Certificate

Now you can sign all certificates generated by [step 1](https://docs.cloudera.com/documentation/enterprise/latest/topics/kafka_security.html#generating_keys_and_certificates) with the CA generated in [step 2](https://docs.cloudera.com/documentation/enterprise/latest/topics/kafka_security.html#creating_your_own_certificate_authority).

1. Create a certificate request from the keystore:

keytool -keystore server.keystore.jks -alias localhost -certreq -file cert-file

where:

* + keystore: the location of the keystore
  + cert-file: the exported, unsigned certificate of the server

1. Sign the resulting certificate with the CA (in the real world, this can be done using a real CA):

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

where:

* + ca-cert: the certificate of the CA
  + ca-key: the private key of the CA
  + cert-signed: the signed certificate of the server
  + ca-password: the passphrase of the CA

1. Import both the certificate of the CA and the signed certificate into the keystore:
2. keytool -keystore server.keystore.jks -alias CARoot -import -file ca-cert

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

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

#!/bin/bash

#Step 1

keytool -keystore server.keystore.jks -alias localhost -validity 365 -genkey

#Step 2

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

keytool -keystore server.truststore.jks -alias CARoot -import -file ca-cert

keytool -keystore client.truststore.jks -alias CARoot -import -file ca-cert

#Step 3

keytool -keystore server.keystore.jks -alias localhost -certreq -file cert-file

openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days 365 -CAcreateserial -passin pass:SamplePassword123

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

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

### Step 4: Configuring Kafka Brokers

Kafka Brokers support listening for connections on multiple ports. If SSL is enabled for inter-broker communication (see below for how to enable it), both PLAINTEXT and SSL ports are required.

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

1. In Cloudera Manager, go to **Kafka** > **Instances**.
2. Go to **Kafka Broker** > **Configurations**.
3. In the **Kafka Broker Advanced Configuration Snippet (Safety Valve) for Kafka Properties**, enter the following information:
4. listeners=PLAINTEXT://*kafka-broker-host-name*:9092,SSL://*kafka-broker-host-name*:9093
5. advertised.listeners=PLAINTEXT://*kafka-broker-host-name*:9092,SSL://*kafka-broker-host-name*:9093

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

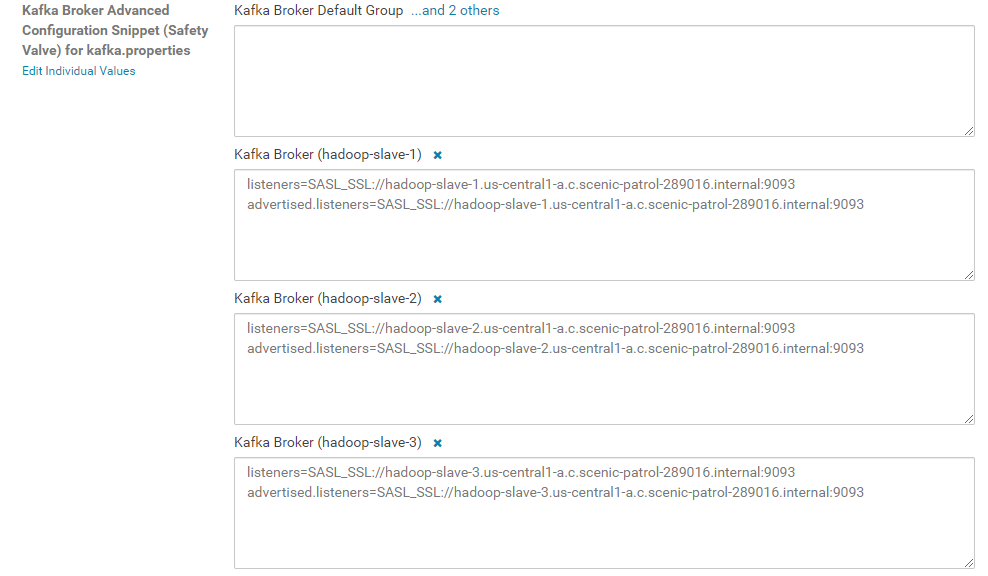
For information about other supported security protocols, see [Using Kafka’s Inter-Broker Security](https://docs.cloudera.com/documentation/enterprise/latest/topics/kafka_security.html#inter-broker-security).

1. Repeat the previous step for each broker.

The advertised.listeners configuration is needed to connect the brokers from external clients.

1. Deploy the above client configurations and rolling restart the Kafka service from Cloudera Manager.

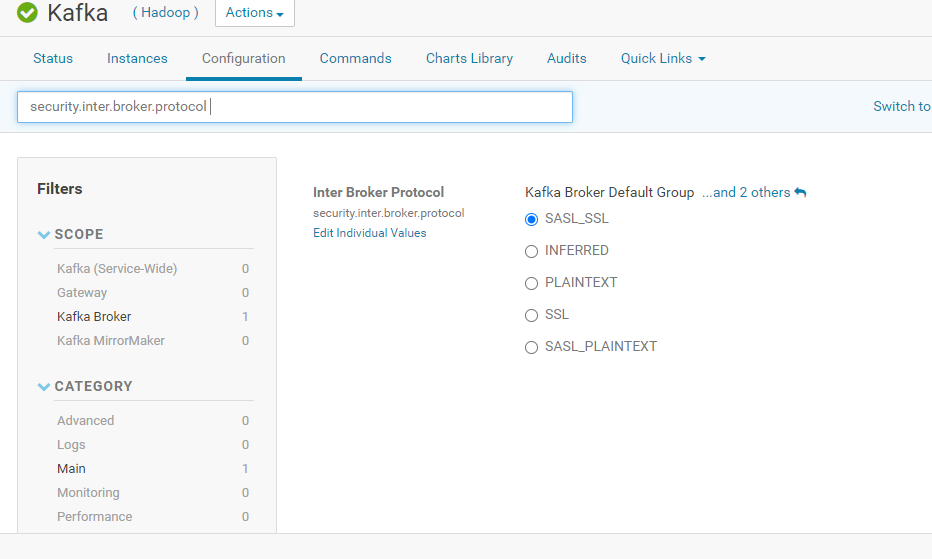
Since we are using Kerberos and Kerberos both ,so our kafka.properties file will look like below:



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

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

Here in our case it will be SASL\_SSL:



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

For instructions, see [Changing the Configuration of a Service or Role Instance](https://docs.cloudera.com/documentation/enterprise/latest/topics/cm_mc_mod_configs.html#cmug_topic_5_3_1).

ssl.keystore.location= /opt/hadoop/security/jks/keystore.jks

ssl.keystore.password=password

ssl.key.password=password

ssl.truststore.location= /opt/hadoop/security/jks/truststore.jks

ssl.truststore.password=password

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

* ssl.client.auth=none: Other options for client authentication are required, or requested, where clients without certificates can still connect. The use of requested is discouraged, as it provides a false sense of security and misconfigured clients can still connect.
* ssl.cipher.suites: A cipher suite is a named combination of authentication, encryption, MAC, and a key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol. This list is empty by default.
* ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1: Provide a list of SSL protocols that your brokers accept from clients.
* ssl.keystore.type=JKS
* ssl.truststore.type=JKS

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

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

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

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

openssl s\_client -debug -connect localhost:9093 -tls1

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

**Note:** ssl.enabled.protocols should include TLSv1.

The output of this command should show the server certificate:

-----BEGIN CERTIFICATE-----

{variable sized random bytes}

-----END CERTIFICATE-----

subject=/C=US/ST=CA/L=Palo Alto/O=org/OU=org/CN=Franz Kafka

issuer=/C=US/ST=CA/L=Palo Alto

/O=org/OU=org/CN=kafka/emailAddress=kafka@your-domain.com

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

### Step 5: Configuring Kafka Clients

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

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

security.protocol=SSL

ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks

ssl.truststore.password=SamplePassword123

If client authentication is required, a keystore must be created as in [step 1](https://docs.cloudera.com/documentation/enterprise/latest/topics/kafka_security.html#generating_keys_and_certificates), it needs to be signed by the CA as in [step 3](https://docs.cloudera.com/documentation/enterprise/latest/topics/kafka_security.html#signing_the_certificate), and you must also configure the following properties:

ssl.keystore.location=/var/private/ssl/kafka.client.keystore.jks

ssl.keystore.password=SamplePassword123

ssl.key.password=SamplePassword123

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

* ssl.provider (Optional). The name of the security provider used for SSL connections. Default is the default security provider of the JVM.
* ssl.cipher.suites (Optional). A cipher suite is a named combination of authentication, encryption, MAC, and a key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol.
* ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1. This property should list at least one of the protocols configured on the broker side.
* ssl.truststore.type=JKS
* ssl.keystore.type=JKS

\*we will create a “client.config” file and add above config into that:

[Devendra@hadoop-slave-1 ~]$ ls

certs.tar **client.config** createsymlinks.sh jaas.conf jdk-8u251-linux-x64.rpm nohup.out opt

[Devendra@hadoop-slave-1 ~]$

[Devendra@hadoop-slave-1 ~]$ cat client.config

**ssl.truststore.location=/opt/hadoop/security/jks/truststore.jks**

**ssl.truststore.password=password**

**ssl.keystore.location=/opt/hadoop/security/jks/keystore.jks**

**ssl.keystore.password=password**

**ssl.key.password=password**

## **Using Kafka’s Inter-Broker Security**

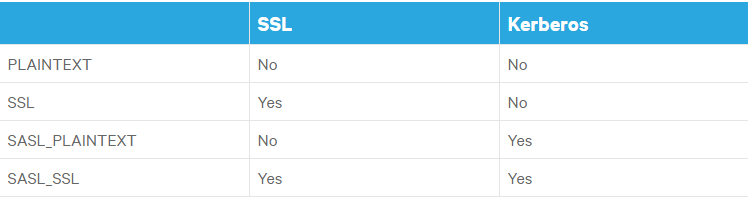
Kafka can expose multiple communication endpoints, each supporting a different protocol. Supporting multiple communication endpoints enables you to use different communication protocols for client-to-broker communications and broker-to-broker communications. Set the Kafka inter-broker communication protocol using the security.inter.broker.protocol property. Use this property primarily for the following scenarios:

* Enabling SSL encryption for client-broker communication but keeping broker-broker communication as PLAINTEXT. Because SSL has performance overhead, you might want to keep inter-broker communication as PLAINTEXT if your Kafka brokers are behind a firewall and not susceptible to network snooping.
* Migrating from a non-secure Kafka configuration to a secure Kafka configuration without requiring downtime. Use a rolling restart and keep security.inter.broker.protocol set to a protocol that is supported by all brokers until all brokers are updated to support the new protocol.

For example, if you have a Kafka cluster that needs to be configured to enable Kerberos without downtime, follow these steps:

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

Kafka 2.0 and higher supports the combinations of protocols listed here.



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

## **Enabling Kerberos Authentication**

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

If you already have a Kerberos server, you can add Kafka to your current configuration. If you do not have a Kerberos server, install it before proceeding. See [Enabling Kerberos Authentication for CDH](https://docs.cloudera.com/documentation/enterprise/latest/topics/cm_sg_intro_kerb.html#xd_583c10bfdbd326ba--6eed2fb8-14349d04bee--76dd).

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

To enable Kerberos authentication for Kafka:

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

To use cached Kerberos credentials, where you use kinit first, use this configuration.

KafkaClient {

com.sun.security.auth.module.Krb5LoginModule required

useTicketCache=true;

};

If you use a keytab, use this configuration. To generate keytabs, see [Step 6: Get or Create a Kerberos Principal for Each User Account](https://docs.cloudera.com/documentation/enterprise/latest/topics/cm_sg_s6_user_principals.html#xd_583c10bfdbd326ba--6eed2fb8-14349d04bee--76e0)).

KafkaClient {

com.sun.security.auth.module.Krb5LoginModule required

useKeyTab=true

keyTab="/etc/security/keytabs/mykafkaclient.keytab"

principal="mykafkaclient/clients.hostname.com@EXAMPLE.COM";

};

1. Create the client.config file containing the following properties.
2. security.protocol=SASL\_PLAINTEXT
3. sasl.kerberos.service.name=kafka
4. we can add above Kerberos config into our client.config file,which we have created earlier:
5. **[Devendra@hadoop-slave-1 ~]$ cat client.config**
6. **security.protocol=SASL\_SSL**
7. **sasl.kerberos.service.name=kafka**
8. **ssl.truststore.location=/opt/hadoop/security/jks/truststore.jks**
9. **ssl.truststore.password=password**
10. **ssl.keystore.location=/opt/hadoop/security/jks/keystore.jks**
11. **ssl.keystore.password=password**
12. **ssl.key.password=password**
13. Test with the Kafka console producer and consumer.

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

kinit user

1. Verify that your topic exists.

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

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

[Devendra@hadoop-slave-1 ~]$ kafka-topics --zookeeper hadoop-slave-2.us-central1-a.c.scenic-patrol-289016.internal –list

/10.128.0.33:2181

20/10/16 11:36:02 INFO zookeeper.ClientCnxn: Session establishment complete on server hadoop-slave-2.us-central1-a.c.scenic-patrol-289016.internal/10.128.0.33:2181, sessionid = 0x3753109ce080039, negotiated timeout = 30000

20/10/16 11:36:02 INFO zookeeper.ZooKeeperClient: [ZooKeeperClient] Connected.

\_\_consumer\_offsets

**first\_topic**

20/10/16 11:36:02 INFO zookeeper.ZooKeeperClient: [ZooKeeperClient] Closing.

20/10/16 11:36:02 INFO zookeeper.ZooKeeper: Session: 0x3753109ce080039 closed

20/10/16 11:36:02 INFO zookeeper.ZooKeeperClient: [ZooKeeperClient] Closed.

20/10/16 11:36:02 INFO zookeeper.ClientCnxn: EventThread shut down

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

export KAFKA\_OPTS="-Djava.security.auth.login.config=/home/user/jaas.conf"

To make these config persistant add this into .bash\_profile.

[Devendra@hadoop-slave-1 ~]$ ls

certs.tar client.config createsymlinks.sh jaas.conf jdk-8u251-linux-x64.rpm nohup.out opt

[Devendra@hadoop-slave-1 ~]$

[Devendra@hadoop-slave-1 ~]$ cat .bash\_profile

# .bash\_profile

# Get the aliases and functions

if [ -f ~/.bashrc ]; then

. ~/.bashrc

fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH

export KAFKA\_OPTS="Djava.security.auth.login.config=/home/Devendra/jaas.conf"

1. Run a Kafka console producer.

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

Last login: Fri Oct 16 11:29:44 2020 from 47.9.166.116

[Devendra@hadoop-slave-1 ~]$ kafka-console-producer --broker-list hadoop-slave-1.us-central1-a.c.scenic-patrol-289016.internal:9093,hadoop-slave-2.us-central1-a.c.scenic-patrol-289016.internal:9093,hadoop-slave-3.us-central1-a.c.scenic-patrol-289016.internal:9093 --topic **first\_topic** --producer.config client.config

1. Run a Kafka console consumer.

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

[Devendra@hadoop-slave-1 ~]$ kafka-console-consumer --bootstrap-server hadoop-slave-1.us-central1-a.c.scenic-patrol-289016.internal:9093,hadoop-slave-2.us-central1-a.c.scenic-patrol-289016.internal:9093,hadoop-slave-3.us-central1-a.c.scenic-patrol-289016.internal:9093 --topic first\_topic **--consumer.config client.config** **--from-beginning**

20/10/16 11:32:02 INFO internals.Fetcher: [Consumer clientId=consumer-1, groupId=console-consumer-85181] Resetting offset for partition first\_topic-1 to offset 0.

20/10/16 11:32:02 INFO internals.Fetcher: [Consumer clientId=consumer-1, groupId=console-consumer-85181] Resetting offset for partition first\_topic-2 to offset 0.

20/10/16 11:32:02 INFO internals.Fetcher: [Consumer clientId=consumer-1, groupId=console-consumer-85181] Resetting offset for partition first\_topic-0 to offset 0.

this is new ssl client

hello kafka

hello

hey how are you

how are you kafka,this is kerberos

hello kafka,this is kerberos

how are you

it is great

hello kafka,welocome to cloudera

hello kafka,this is kerberos

**Disable TLS/SSL:**

To disable TLS, the first step is to stop cloudera-scm-server on the server node and `cloudera-scm-agent on every node.

Then we need to replace use\_tls=1 to use\_tls=0 in /etc/cloudera-scm-agent/config.init on every node.

Since cloudera manager server stores its configurations on the mysql database, we need to disable "Use TLS Encryption for Agent" from the database:

[ec2-user@ip-172-20-0-4 ~]$ mysql -u root -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 10342

Server version: 5.6.34-log MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> **use scm;**

Database changed

mysql> **UPDATE CONFIGS SET value='false' where attr="agent\_tls";**

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

\_tls

I would recommend reviewing the Cloudera Manager log for clues, but, for now, access your Cloudera Manager database and run the following:

delete from CONFIGS where ATTR='web\_tls';

This will disable TLS for the CM UI

Afterward, try starting again.

If that doesn't help, let us know.

Exception:

/var/log/cloudera-scm-server/cloudera-scm-server.log:

2019-12-17 17:44:06,619 WARN ActionablesProvider-0:com.cloudera.server.web.cmf.StatusProvider: Failed to submit task for getting status from SERVICE\_MONITORING

com.cloudera.cmon.MgmtServiceNotRunningException: SERVICE\_MONITORING is not running

at com.cloudera.cmon.MgmtServiceLocator.getNozzleIPC(MgmtServiceLocator.java:145)

at com.cloudera.server.web.cmf.StatusProvider$SubjectStatusCustomFuture.<init>(StatusProvider.java:618)

at com.cloudera.server.web.cmf.StatusProvider.getStatus(StatusProvider.java:1043)

at com.cloudera.server.web.cmf.StatusProvider.getRolesStatus(StatusProvider.java:1292)

INFO CMMetricsForwarder-0:com.cloudera.server.cmf.components.ClouderaManagerMetricsForwarder: Failed to send metrics.

java.lang.reflect.UndeclaredThrowableException

org.apache.avro.AvroRemoteException: java.net.ConnectException: Connection refused

/var/log/cloudera-scm-agent/cloudera-scm-agent.log:

[root@hadoop-slave-1 jks]# tail -f /var/log/cloudera-scm-agent/cloudera-scm-agent.log

Traceback (most recent call last):

File "/usr/lib64/cmf/agent/build/env/lib/python2.6/site-packages/cmf-5.14.1-py2.6.egg/cmf/monitor/firehose.py", line 116, in \_send

self.\_port)

File "/usr/lib64/cmf/agent/build/env/lib/python2.6/site-packages/avro-1.6.3-py2.6.egg/avro/ipc.py", line 469, in \_\_init\_\_

self.conn.connect()

File "/usr/lib64/python2.6/httplib.py", line 771, in connect

self.timeout)

File "/usr/lib64/python2.6/socket.py", line 567, in create\_connection

raise error, msg

error: [Errno 111] Connection refused

Solution:

Re: After disabling TLS/SSL ,cloudera management services are not able to start

Hi [@pdev](https://community.cloudera.com/t5/user/viewprofilepage/user-id/69852) ,

You may want to check this community thread:

[https://community.cloudera.com/t5/Support-Questions/how-to-rollback-cloudera-manager-tls-configurati...](https://community.cloudera.com/t5/Support-Questions/how-to-rollback-cloudera-manager-tls-configuration-without/m-p/46484)

It is possible that some of the data was still in CM database which expects TLS login. Could you please run below SQL command to confirm?

select attr, value from CONFIGS where attr in ('web\_tls', 'agent\_tls');

If you see any returned value is "true" then you need to update the CM database manually. See below steps:

1. Back up database/table before you makde any further changes
2. Run the following queries in the CM database:

*Update TLS for web\_tls update CONFIGS set value = 'false' where attr = 'web\_tls';  
Update TLS for agent\_tls update CONFIGS set value = 'false' where attr = 'agent\_tls';*

1. In the browser do clear cache including passwords, and cookies before trying to login.

Thanks and hope this helps!

Li Wang, Technical Resolution Manager

***Was your question answered? Make sure to mark the answer as the accepted solution.***  
***If you find a reply useful, say thanks by clicking on the thumbs up button.***