# Web Services and Applications Deployment Guide

## Transport Layer Security (TLS)

5/7/2025

# Transport Layer Security (TLS)

## Contents

# Configuring TLS between Web Services and Configuration Server

Web Services can use a secured Transport Layer Security (TLS) connection mechanism to connect to Configuration Server. When configured, Web Services connects to a secure port on Configuration Server, verifies the server's authority, and encrypts/decrypts network traffic. You can configure secured connections to Configuration Server in the following ways:

- Minimal configuration
- Validate the certificate against the CA

## Prerequisites

Before configuring Web Services, make sure the Configuration Server secure port is configured as described in Introduction to Genesys Transport Layer Security in the *Genesys Security Deployment Guide* and that all certificates for server host and the certificate authority are configured and available.

## Minimal configuration

Web Services does not check the server's certificate against the Certificate Authority, but all traffic is encrypted. To configure Web Services with minimal configuration, all you need to do is configure a connection to a secured port on Configuration Server. You can do this using **either** of the following methods:

- For the initial connection to Configuration Server, set the tlsEnabled option to `true` in the onPremiseSettings section of the **application.yaml** file (**onpremise-settings.yaml** file if you are installing Web Services and Applications version 8.5.201.09 or earlier). This creates a secured connection to Configuration Server the first time Web Services starts.
- For an environment that is already configured with Configuration Manager synchronization enabled, you can make changes with Configuration Manager as described in the Genesys Security Deployment Guide. These changes are synchronized back to the Cassandra database from Configuration Manager.

## Validate the certificate against the CA

In order to support the client-side certificate check, Web Services needs the public key for the Certificate Authority (CA). Web Services supports the PEM and JKS key storage formats, but recommends using JKS because it's compatible with both Cassandra and HTTPS.

Complete the steps below to validate the certificate against the CA.

> **Important**
>
> The steps described in this procedure are meant to be an example for developers and should not be used in production. For a production environment, you should follow

> your own company's security policies for creating and signing certificates.

**Start**

1. If you plan to use a JKS file, you can generate if from a PEM file by importing the PEM certificate, as shown here:

   ```
   keytool -importcert -file ca_cert.pem -keystore ca_cert.jks
   ```

2. Once you have the **ca_cert.jks** file, place it in a location available from your Web Services host, such as:
   - A local folder on the Web Services host
   - A network share
3. Configure the following options in the serverSettings section of the **application.yaml** file (**server-settings.yaml** if you're installing Web Services and Applications version 8.5.201.09 or earlier):
   - For a PEM file, set caCertificate to the location of the file. For example:

     ```
     caCertificate: /opt/ca_cert.pem
     ```

   - For a JKS file, set caCertificate to the location of the file and set jksPassword to the password for the key storage. For example:

     ```
     caCertificate: /opt/ca_cert.jks
     jksPassword: pa$$word
     ```

**End**

# Configuring TLS for connections with Cassandra

Genesys supports Transport Layer Security (TLS) for connections from Web Services to Cassandra and between Cassandra nodes. You can configure secured connections for one or both of the following scenarios:

- Secure connections from Web Services to Cassandra
- Secure connections between Cassandra nodes

## Secure Connections from Web Services to Cassandra

**Prerequisites**

- You have installed Bash, Java keytool and OpenSSL

Complete the following steps to configure TLS for connections from Web Services to Cassandra.

> **Important**
>
> The steps described in this procedure are meant to be an example for developers and should not be used in production. For a production environment, you should follow your own company's security policies for creating and signing certificates.

**Start of procedure**

1. Do one of the following:
   - Create the server-side keystore with a self-signed certificate and the client-side truststore — which contains the public part of server certificate. Run the following commands:

     ```
     #!/bin/bash
     #generate keypair
     keytool -genkeypair -alias cassandra -keyalg RSA -keysize 1024 -dname "CN=cassandra,
     OU=Test, O=Test Ltd, C=US" -keystore server.jks
     -storepass password -keypass password

     #export certificate
     keytool -exportcert -alias cassandra -file client.pem -keystore server.jks -storepass
     password -rfc

     #create client truststore and import certificate
     keytool -importcert -alias cassandra -file client.pem -keystore client.jks -storepass
     password -noprompt
     ```

   - Create a self-signed root authority, use it to sign the server certificate, store it to **server.jks** and create the client-side truststore, which trusts all certificates signed with root authority. Run the following commands:

     ```
     #!/bin/sh

     #generate self-signed root certificate
     keytool -genkeypair -alias root -keyalg RSA -keysize 1024 -validity 3650 -dname
     "CN=TestRoot, OU=Dev, O=Company, C=US" -keystore root.jks
     -storepass password -keypass password

     #export root certificate
     keytool -exportcert -alias root -file root.crt -keystore root.jks -storepass password

     #generate server-side certificate
     keytool -genkeypair -alias server -keyalg RSA -keysize 1024 -validity 3650 -dname
     "CN=TestServer, OU=Dev, O=Company, C=US"
     -keystore server.jks -storepass password -keypass password

     #create the sign request for server certificate
     keytool -certreq -alias server -keystore server.jks -file server.csr -storepass
     password -keypass password

     #export private key of root auth: need later for signing the server certificate
     keytool -v -importkeystore -srckeystore root.jks -srcalias root -destkeystore
     root.p12 -deststoretype PKCS12 -noprompt
     -destkeypass password -srckeypass password -destalias root -srcstorepass password
     -deststorepass password

     openssl pkcs12 -in root.p12 -out private.pem -password pass:password -passin
     pass:password -passout pass:password
     ```

```
rm root.p12

#sign the certificate
openssl  x509  -req  -CA   private.pem  -in server.csr -out server.crt  -days 3650
-CAcreateserial -passin pass:password
rm private.pem
rm private.srl
rm server.csr

#import root certificate to client side trust store
keytool -importcert -alias root -file root.crt -keystore client.jks -storepass
password -noprompt

#import root certificate to server side key store
keytool -importcert -alias root -file root.crt -keystore server.jks -storepass
password -noprompt
rm root.crt

#import certificate sign reply into server-side keystore
keytool -import -trustcacerts -alias server -file server.crt -keystore server.jks
-storepass password -keypass password
rm server.crt
```

2. Configure Cassandra to use your generated certificates for the client connection by setting the `client_encryption_options` in the **cassandra.yaml** file. For example:

```
client_encryption_options:
     enabled: true
     keystore: <absolute path to server.jks file>
     keystore_password: password
     #the password specified in while creating storage
     # For the purpose of the demo the default settings were used.
     # More advanced defaults below:
     #protocol: TLS
     #algorithm: SunX509
     #store_type: JKS
     #cipher_suites: [TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA]
```

> **Important**
>
> To enable support for encryption, you must have the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction installed.

3. Confirm the Cassandra nodes can start successfully:
   a. Edit the **conf/log4j-server.properties** file and uncomment the following line:

      ```
      log4j.logger.org.apache.cassandra=DEBUG
      ```

   b. Start Cassandra and check the logs. If the configuration was successful, you shouldn't see any errors.
4. Check that SSL-to-client is working successfully using cassandra-cli:
   a. Confirm that unsecured connections aren't possible by starting cassandra-cli locally — this forces it to connect to the Cassandra instance running on localhost. You should expect to see the exception in the cassandra-cli output.
   b. Confirm that secured connections are possible by running the following command from the directory

where cassandra-cli is installed:

```
./cassandra-cli -tf org.apache.cassandra.cli.transport.SSLTransportFactory -ts
<absolute path to client truststore cass_client.jks> -tspw somePassword
```

c. If the configuration is successful, you should see the "Connected to" welcome message:

```
Connected to: "Test Cluster" on 127.0.0.1/9160
Welcome to Cassandra CLI version 1.2.12
```

5. Configure Web Services to enable secure connections:
   a. On the Web Services node, open the **application.yaml** file.
   b. In the `cassandraCluster` section, configure the following settings:

| Parameter | Type | Default | Description |
|---|---|---|---|
| useSSL | Boolean | `false` | Specify `true` to connect Cassandra to an SSL channel. This paramater only applies to Cassandra 1.2.x. |
| truststore | String | | Specify the absolute path to the trustore file. Ensure that Web Services can read the file. The supported format is **JKS**. |
| truststorePassword | String | | Specify the Truststore password. |
| sslProtocol | String | TLS | Specify the SSL protocol to be used. This protocol is passed to JAVA security. |
| cipherSuites | String | `[TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA]` | Specify a list of ciphers in the form of a yaml list. The list must match the list that is configured in Cassandra. |

   c. Specify your Cassandra version using the cassandraVersion parameter.

> **Important**
> Web Services does not support Cassandra 1.1 for on premise deployments.

**End or procedure**

## Secure connections between Cassandra nodes

When you enable SSL for connections between Cassandra nodes, you ensure that communication between nodes in the Cassandra cluster is encrypted, and that only other authorized Cassandra nodes can join the cluster.

The steps below show you how to create a single certificate to be used by all Cassandra nodes in the cluster. This simplifies cluster management because you don't need to generate a new certificate each time you add a new node to the cluster, which means you don't need to restart all nodes to load the new certificate.

> **Important**
>
> The steps described in this procedure are meant to be an example for developers and should not be used in production. For a production environment, you should follow your own company's security policies for creating and signing certificates.

**Start**

1. Generate a keystore and truststore. See Step 1 of Secure connections from Web Services to Cassandra for details.
2. On each Cassandra node in the cluster, set `server_encryption_options` in the **cassandra.yaml** file. For example:

```
server_encryption_options:
     internode_encryption: all
     keystore: <absolute path to keystore >
     keystore_password: <keystore password - somePassword in our sample>
     truststore: <absolute path to trustore>
     truststore_password: <truststore password - somePassword in our sample>
```

3. Check the Cassandra logs. If the configuration was successful, you shouldn't see any errors.

**End**

# Next Step

- Back to Configuring security