Deployment Guide

Cassandra Security

12/16/2025

Cassandra Security

Unauthorized access to Cassandra data is possible at several points:

- Direct access via "standard" interfaces: Thrift and CQL
- Access to data traveling through the network
- Access to data files that Cassandra stores on hard drives

Cassandra's default configuration provides mechanisms to secure direct interfaces (through authentication and authorization) and network traffic (through the use of TLS). The data stored on hard drives can be secured either by third-party commercial offerings or with some development investments.

The following sections describe how to provide secure access to:

- External Cassandra

External Cassandra

Securing access interfaces

You can secure your access interfaces based on an authentication and authorization scheme. In other words, Cassandra needs to know:

- Who is trying to access the system
- Whether they are allowed to access the system at all
- If so, which data they should have access to

With the default setup, anybody is allowed to access all the data.

Authentication

Authentication (who) is managed by the **authenticator** parameter in the **cassandra.yaml** file. By default, GWE 8.5 only provides a login/password authentication scheme with the use of **PasswordAuthenticator**.

Procedure

Start

1. Edit ***Cassandra installation directory*/conf/cassandra.yaml**:

1. Change the **authenticator** parameter to `PasswordAuthenticator`. Note that by default, the **authenticator** option is set to `AllowAllAuthenticator`.
2. Tune your **system_auth** keyspace replication according to the DataStax system_auth documentation.

2. Restart the Cassandra client.
   The default superuser name and password that you use to start the client is stored in Cassandra:

   ```
   <client startup string> -u cassandra -p cassandra
   ```

3. Start cqlsh using the superuser name and password (cassandra):
   ```
   ./cqlsh -u cassandra -p cassandra
   ```
4. Deactivate the default superuser, which is called `cassandra`, as mentioned above. This step is optional but highly recommended.
   1. Create a superuser with another name.
   2. Log in as the newly create superuser.
   3. Change the `cassandra` user password to something long and incomprehensible, and then forget about it. It won't be used again.
   4. Take away the `cassandra` user's superuser status.
5. Use the following CQL 3 statements to set up user accounts and then grant permissions to access the database objects:
   - CREATE USER
   - GRANT
6. Set the new user name and password to the values of the [cassandraKeyspace] userName and password options for the Web Engagement Cluster application.

End

Authorization

Authorization (which data) is managed by the authorizer in the GWE cluster configuration options. Cassandra offers a familiar relational database **GRANT/REVOKE** paradigm to grant or revoke permissions for accessing data.

Procedure

Start

Edit ***Cassandra installation directory*/conf/cassandra.yaml**:

1. Change the **authorizer** parameter to `CassandraAuthorizer`. Note that by default, the **authorizer** option is set to `AllowAllAuthorizer`.
2. Tune your **system_auth** keyspace replication according to the DataStax system_auth documentation. Note that the validity period for permissions caching is 2000 ms.

For more information about permissions see the DataStax Object permissions documentation.

End

CQL supports these authorization statements:

- GRANT
- LIST PERMISSIONS
- REVOKE

Resource Access Point Configuration

Prerequisites

- The Cassandra Resource Access Point applications are created and configured

Procedure

Start

For all Cassandra Resource Access Points:

1. Open the **cassandraClient** (previously **cluster**) configuration option section.
2. Set the **userName** option to the name of an already-created user.
3. Set the **password** option to the user's password.

End

Securing Network Traffic

The client-to-node and node-to-node traffic in your Cassandra deployment may require protection. They can both be secured by using SSL (Secure Sockets Layer) encryption.

Client-to-Node Encryption

Client-to-node encryption uses SSL to protect data that is traveling from client machines to a database cluster. It does this by establishing a secure channel between the client and the coordinator node.

Prerequisites

- You must install Java Cryptography Extension (to enable 256-bit encryption).
- All nodes must have all of the relevant SSL certificates. See Preparing server certificates.

Procedure

Start

On each node that belongs to the Cassandra cluster, edit ***Cassandra installation directory*/conf/cassandra.yaml**:

1. Set the client_encryption_options/enabled option to `true`.
2. Set the appropriate paths to your .keystore and .truststore files in the client_encryption_options/keystore and client_encryption_options/truststore options.
3. Provide the required passwords in client_encryption_options/keystore_password and client_encryption_options/truststore_password. The passwords must match the passwords used when generating the keystore and the truststore.
4. To enable client certificate authentication, set the client_encryption_options/require_client_auth option to `true`.

End

Node-to-Node Encryption

Node-to-node encryption uses SSL to protect data being transferred between cluster nodes. This includes node-to-node gossip communication.

Prerequisites

- You must install Java Cryptography Extension (to enable 256-bit encryption).
- All nodes must have all of the relevant SSL certificates. See Preparing server certificates.

Procedure

Start

On each node that belongs to the Cassandra cluster, edit ***Cassandra installation directory*/conf/cassandra.yaml**:

1. Set the server_encryption_options/internode_encryption option to one of the following:
   - **all**—Cassandra encrypts all internodal traffic
   - **dc**—Cassandra encrypts all traffic between datacenters
   - **rack**—Cassandra encrypts all traffic between racks
2. Set the appropriate paths to your .keystore and .truststore files in the server_encryption_options/keystore and server_encryption_options/truststore options.
3. Provide the required passwords in server_encryption_options/keystore_password and server_encryption_options/truststore_password. The passwords must match the passwords used when generating the keystore and the truststore.
4. To enable client certificate authentication, set the server_encryption_options/require_client_auth option to `true`.

End

Resource Access Point Configuration

Prerequisites

- The Cassandra Resource Access Point applications are created and configured.

Procedure

Start

For all Cassandra Resource Access Points:

1. Open the properties for the port with an ID of `native`.
2. Set this port to `secured`.
3. If your Cassandra cluster has its own properly configured security certificate, enable Mutual TLS for this port.

End

Web Engagement Application Configuration

Prerequisites

- The Web Engagement cluster applications are created and configured.
- The Web Engagement cluster applications have their own properly configured security certificates.

Procedure

Start

- For every Web Engagement application:
  1. Make sure its security certificate was correctly configured.
  2. For security fine tuning, read the description of the TLS options in Configuration Options Reference Manual.
- For every Web Engagement cluster application:
  - Set the Web Engagement application port to `secured`.

End

Using cqlsh

If you are planning to use the cqlsh standard utility with encryption, please consult the Datastax

documentation.

Securing access interfaces

You can secure your access interfaces based on an authentication and authorization scheme. In other words, Cassandra needs to know:

- Who is trying to access the system
- Whether they are allowed to access the system at all
- If so, which data they should have access to

With the default setup, anybody is allowed to access all the data.

Authentication

Authentication (who) is managed by the **authenticator** parameter in the **cassandra.yaml** file. By default, GWE 8.5 only provides a login/password authentication scheme with the use of **PasswordAuthenticator**.

Procedure

Start

1. Change the authenticator option to `PasswordAuthenticator`.
   By default, the **authenticator** option is set to `AllowAllAuthenticator`.
2. Tune your **system_auth** keyspace replication according to the DataStax system_auth documentation.
3. Restart the Cassandra client.
   The default superuser name and password that you use to start the client is stored in Cassandra:

   ```
   <client startup string> -u cassandra -p cassandra
   ```
4. Start cqlsh using the superuser name and password (cassandra):
   ```
   ./cqlsh -u cassandra -p cassandra
   ```
5. Deactivate the default superuser, which is called `cassandra`, as mentioned above. This step is optional but highly recommended.
   1. Create a superuser with another name.
   2. Log in as the newly create superuser.
   3. Change the `cassandra` user password to something long and incomprehensible, and then forget about it. It won't be used again.
   4. Take away the `cassandra` user's superuser status.
6. Use the following CQL 3 statements to set up user accounts and then grant permissions to access the database objects:
   - CREATE USER
   - GRANT

7. Set the new user name and password to the values of the [cassandraKeyspace] userName and password options for the Web Engagement Cluster application.

End

Authorization

Authorization (which data) is managed by the authorizer in the GWE cluster configuration options. Cassandra offers a familiar relational database **GRANT/REVOKE** paradigm to grant or revoke permissions for accessing data.

Procedure

Start

1. Change the authorizer|authorizer option in the cassandra.yaml file to `CassandraAuthorizer`.
   By default, the **authorizer** option is set to `AllowAllAuthorizer`.
2. Tune your **system_auth** keyspace replication according to the DataStax system_auth documentation.
   Note that the validity period for permissions caching is 2000 ms.
   For more information about permissions see the DataStax Object permissions documentation.

End

CQL supports these authorization statements:

- GRANT
- LIST PERMISSIONS
- REVOKE

Securing Network Traffic

The client-to-node and node-to-node traffic in your Cassandra deployment may require protection. They can both be secured by using SSL (Secure Sockets Layer) encryption.

Client-to-Node Encryption

Client-to-node encryption uses SSL to protect data that is traveling from client machines to a database cluster. It does this by establishing a secure channel between the client and the coordinator node.

Prerequisites

- You must install Java Cryptography Extension (to enable 256-bit encryption).
- All nodes must have all of the relevant SSL certificates. See Preparing server certificates.
- To enable client-to-node SSL, you must set the *client_encryption_options* options in the cassandra.yaml file.

- All Web Engagement node applications must have correctly configured security certificates. To learn more about how to configure your node application certificates, read the description of the TLS options in Configuration Options Reference Manual.

Procedure

Start

For each node in the Cassandra cluster, open the cassandra.yaml file and:

1. Set the client_encryption_options\enabled option to `true`.
2. Set the appropriate paths to your .keystore and .truststore files in the client_encryption_options\keystore and client_encryption_options\truststore options.
3. Provide the required passwords in client_encryption_options\keystorePassword and client_encryption_options\truststorePassword. The passwords must match the passwords used when generating the keystore and the truststore.
4. To enable client certificate authentication, set the client_encryption_options\require_client_auth option to `true`.
5. For security fine tuning, read the description of the TLS options in Configuration Options Reference Manual.

End

Node-to-Node Encryption

Node-to-node encryption uses SSL to protect data being transferred between cluster nodes. This includes node-to-node gossip communication.

Prerequisites

- You must install Java Cryptography Extension (to enable 256-bit encryption).
- All nodes must have all of the relevant SSL certificates. See Preparing server certificates.
- To enable node-to-node SSL, you must set the *server_encryption_options* options in the cassandra.yaml file.

Procedure

Start

On each node that belongs to the Cassandra cluster:

1. Set the server_encryption_options\internode_encryption option to one of the following:
   - **all**—Cassandra encrypts all internodal traffic
   - **dc**—Cassandra encrypts all traffic between datacenters

- **rack**—Cassandra encrypts all traffic between racks

2. Set the appropriate paths to your .keystore and .truststore files in the server_encryption_options\keystore and server_encryption_options\truststore options.
3. Provide the required passwords in server_encryption_options\keystore_password and server_encryption_options\truststore_password. The passwords must match the passwords used when generating the keystore and the truststore.
4. To enable client certificate authentication, set the server_encryption_options\require_client_auth option to `true`.

End

Using cqlsh

If you are planning to use the cqlsh standard utility with encryption, please consult the Datastax documentation.