Cloudera Data Warehouse Private Cloud 1.5.1

Securing CDW on Private Cloud

Date published: 2020-08-17
Date modified: 2023-06-13

CLOUDERA

https://docs.cloudera.com/

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Authenticating users in CDW Private Cloud

Cloudera Data Warehouse (CDW) supports LDAP and Kerberos for authenticating users to access databases and tables from Business Intelligence (BI) clients and tools such as Hue, Beeline, Impyla, Impala-shell, and other JDBC clients.

Authentication using LDAP

If you choose to use LDAP to authenticate users to connect to a Virtual Warehouse from a client shell such as Beeline, Impyla, Impala-shell, and so on, then you must create Bind users. However, the Bind users must either specify the username and password inline with the command or after submitting the command from the client.

CDW does not support LDAP and Kerberos simultaneously for connecting to Hive Virtual Warehouse. You have the option to select either LDAP or Kerberos for authenticating users to Hive Virtual Warehouses while creating a new Virtual Warehouse. The default authentication mode is LDAP. There is no such restriction for Impala. However, if you enable the Unified Analytics mode on Impala Virtual Warehouses, then certain Hive components are also included in the Impala Virtual Warehouse, therefore, the authentication mode restriction applies to these Hive components as well. You must specify the authentication mode for Unified Analytics components when creating an Impala Virtual Warehouse. The default authentication mode for the Hive components in the Unified Analytics mode is LDAP. The authentication mode that you set here applies only to the Unified Analytics' components (mainly Hive). The Impala components continue to support both authentication modes. If you connect to the Impala Virtual Warehouse remotely, then both LDAP and Kerberos authentication modes can be used. But if you connect to the Impala Virtual Warehouse in the Unified Analytics mode, then only the selected authentication mode is used.

Authentication using Kerberos

Kerberos uses passwords stored in the Kerberos keytab files. This removes the need to specify username and password as parameters inline with the command or after submitting the command from a JDBC client to connect to a Virtual Warehouse in CDW, while adding a layer of security.

Important:

- The Private Cloud Base cluster must be Kerberized to use Kerberos as the authentication mode for CDW.
- CDW requires all user Kerberos principals to be present in the configured LDAP server as well.

Impala can use LDAP or Kerberos without any pre-configuration. To use Kerberos for Hive, you must select KERBEROS from the Hive Authentication Mode drop-down menu while creating a Hive Virtual Warehouse.

How predefined Kerberos principals are used in CDW Private Cloud

By default, Cloudera Data Warehouse (CDW) creates Kerberos principal names for Database Catalogs and Environments using the service hostname and the deterministic namespace name based on the name of the Database Catalog or Environment when you create a Database Catalog or an Environment. However, you can generate and provide the keytabs, if needed.

The service principals for CDW need to be the same as on the base cluster. For more information, see Customizing Kerberos principals in the CDP Private Base documentation.

By default, the host principals are generated programmatically. You can generate and provide the keytabs, but the hostnames in the Kerberos principals are fixed. CDW uses a deterministic namespace and environment IDs for the Kerberos principals.

When you specify an Environment or Database Catalog name, CDW appends a prefix as shown in the following table, as well as the Kerberos principal name based on them:

CDW entity	User-specified name	Namespace IDs with CDW-assigned prefix	Hive Kerberos principal name
Environment	my-test-env	env-my-test-env-default	hive/dwx-env-my-test-env.cdp.local@REALM.EXAMPLE.COM
Database Catalog	my-test-catalog	warehouse-my-test-catalog	hive/metastore-service.warehouse-warehouse-my-test-catalog.svc.cluster.local@REALM.EXAMPLE.COM
Virtual Warehouse	my-impala-warehouse	impala-my-impala-warehouse	NA

Note: The length of the namespace ID after CDW applies a prefix to the Environment or Database Catalog name, including the hyphen (-), should not exceed 63 characters. You can specify an Environment name 35 characters long and Database Catalog 53 characters long.

When using FreeIPA, the environment name can be maximum 17 characters long.

What is delegation username in CDW Private Clouds

You must specify a delegation username and password to impersonate authorization requests from Hue to the Impala engine during environment activation. The delegation user and password can authenticate users through an LDAP service account.

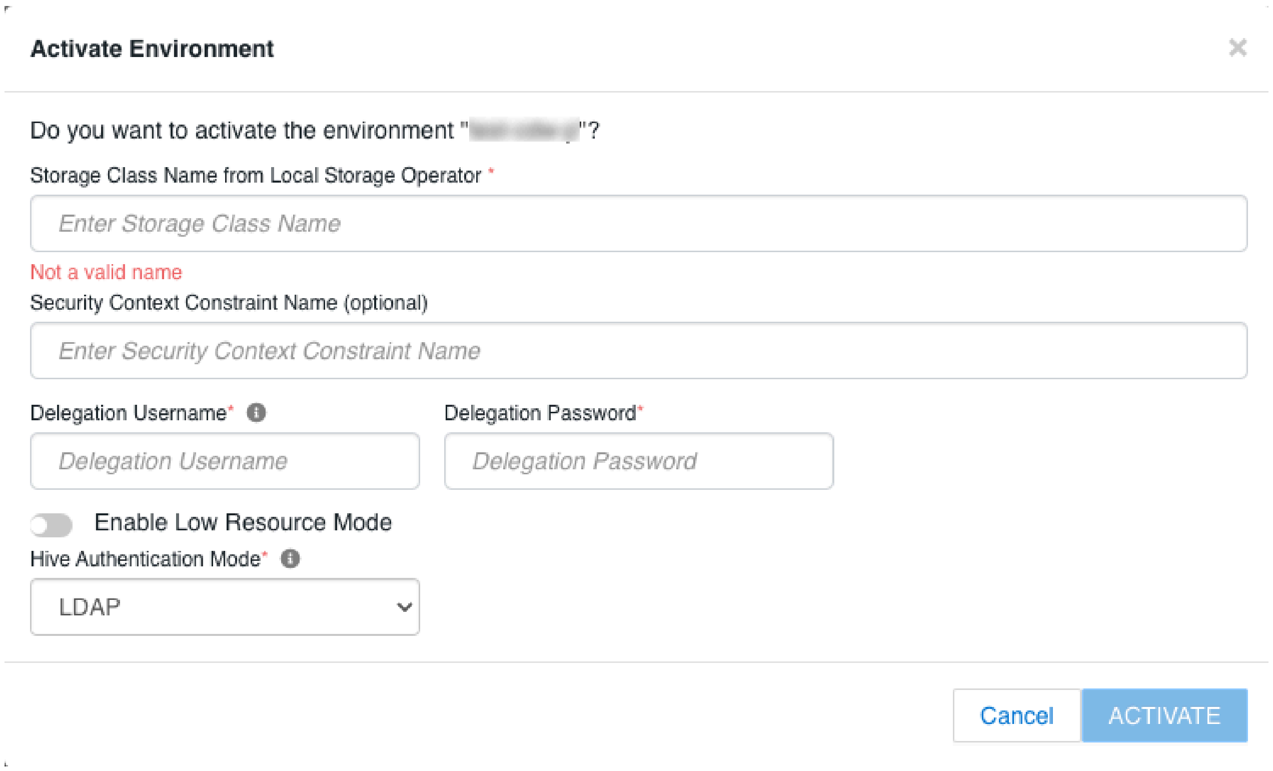
The ability to specify an LDAP delegation user also allows you to freely use special characters in your LDAP Bind DN, as CDW no longer has to inherit and process the delegation user from the LDAP Bind DN.

Note:

- The delegation user and the LDAP Bind user configured on the **Administration** page of the Management Console are not necessarily the same user.
- The special characters used in the LDAP Bind user password are not exactly the same as the ones that can be used in the delegation user password, because only the following characters are supported to be used in the LDAP Bind user password: ! # $ % ( ) * + , - . / : ; = ? @ [ ] ^ _ ` { | } ~.
- The following special characters are not supported to be used in the name of the delegation user or in the Distinguished Name of the LDAP Bind user: < > & ' ".

You can change the delegation username and password even after activating the environment.

The following image shows the CDW **Activation Settings** page containing the Delegation Username and Delegation Password fields:



Activate Environment

Do you want to activate the environment "[redacted]"?

Storage Class Name from Local Storage Operator *

Enter Storage Class Name

Not a valid name

Security Context Constraint Name (optional)

Enter Security Context Constraint Name

Delegation Username *

Delegation Username

Delegation Password *

Delegation Password

Enable Low Resource Mode

Hive Authentication Mode *

LDAP

Related Information

Changing delegation username and password

Changing delegation username and password

You specify the delegation username and password while activating an environment. You can change the delegation username or password from the Environment Details page.

Procedure

1. Log in to the Data Warehouse service as a DWAdmin.
2. Go to Environments ⋮ Edit CONFIGURATIONS .
3. Enter a new Delegation Username and/or Delegation Password.

   Note:

   - The delegation user and the LDAP Bind user configured on the **Administration** page of the Management Console are not necessarily the same user.
   - The special characters used in the LDAP Bind user password are not exactly the same as the ones that can be used in the delegation user password, because only the following characters are supported to be used in the LDAP Bind user password: ! # $ % ( ) * + , - . / : ; = ? @ [ ] ^ _ ` { | } ~.
   - The following special characters are not supported to be used in the name of the delegation user or in the Distinguished Name of the LDAP Bind user: < > & ' ".
4. Click Apply Changes.

How to enable SSL for MariaDB, MySQL, and Oracle databases

Cloudera requires that you secure the network connection between the default Database Catalog Hive MetaStore (HMS) in Cloudera Data Warehouse (CDW) and the relational database hosting the base cluster’s HMS using SSL encryption.

You must provide the SSL certificate of the database either by:

- Providing the SSL certificate while installing the Data Services on the Install Private Cloud Data Services on Existing Container Cloud Configure Kubernetes step under the Additional Certificates section. See Installing in an internet environment.
- Importing the SSL certificate to the trust store on the base cluster before installing CDP Private Cloud.

Configuring MySQL database to use SSL for Data Warehouse

SSL encrypts the connection between the MySQL server and the Hive MetaStore (HMS) on the base cluster. You must enable SSL for the MySQL database before setting up the CDP Private Cloud Data Services and add the MySQL root Certificate Authorities (CA) to the Cloudera Manager truststore.

Procedure

1. SSH into the MySQL database host.
2. Start the MySQL server:

```
service mysqld start
```

3. Establish an encrypted connection with the client:

```
mysql -p --ssl-mode=required
```

4. Verify whether SSL is enabled on MySQL by running the following command:

```
mysql> show global variables like '%ssl%';
```

If SSL is enabled, you see the value of have_ssl equal to YES, as follows. Otherwise, you see the value of have _ssl equal to DISABLED:

```
+---------------+----------+
| Variable_name | Value    |
+---------------+----------+
| have_openssl  | YES      |
| have_ssl      | YES      |
| ...           | ...      |
```

If SSL is enabled, then skip to step 11.

5. Create a certificate authority by running the following commands:

```
mkdir /etc/my.cnf.d/ssl/
cd /etc/my.cnf.d/ssl/
openssl genrsa 2048 > ca-key.pem
```

6. Create a certificate for the server using the CA certificate generated earlier by running the following command:

```
openssl req -new -x509 -nodes -days 365000 -key ca-key.pem -out ca-cert.
pem
openssl req -newkey rsa:2048 -days 365 -nodes -keyout server-key.pem -out
 server-req.pem
openssl rsa -in server-key.pem -out server-key.pem
```

7. Create a certificate for the clients using the same CA certificate by running the following command:

```
openssl x509 -req -in server-req.pem -days 365 -CA ca-cert.pem -CAkey ca-
key.pem -set_serial 01 -out server-cert.pem
```

8. Add the following lines in the /etc/my.cnf.d/server.cnf file under the [mysqld] section:

```
ssl-ca=/etc/my.cnf.d/ssl/ca-cert.pem
ssl-cert=/etc/my.cnf.d/ssl/server-cert.pem
ssl-key=/etc/my.cnf.d/ssl/server-key.pem
bind-address=*
```

You can view the content of the server.cnf file by running the following command:

```
vim /etc/my.cnf.d/server.cnf
```

9. Restart the MySQL server:

```
service mysqld restart
```

10. Check the SSL status by running the following commands:

```
mysql -p --ssl-mode=required
> SHOW VARIABLES LIKE '%ssl%';
> status
```

Sample output:

```
> SHOW VARIABLES LIKE '%ssl%';
+-------------------------------------+-----------------+
| Variable_name                       | Value           |
+-------------------------------------+-----------------+
| admin_ssl_ca                        |                 |
| admin_ssl_capath                    |                 |
| admin_ssl_cert                      |                 |
| admin_ssl_cipher                    |                 |
| admin_ssl_crl                       |                 |
| admin_ssl_crlpath                   |                 |
| admin_ssl_key                       |                 |
| have_openssl                        | YES             |
| have_ssl                            | YES             |
| mysqlx_ssl_ca                       |                 |
| mysqlx_ssl_capath                   |                 |
| mysqlx_ssl_cert                     |                 |
| mysqlx_ssl_cipher                   |                 |
| mysqlx_ssl_crl                      |                 |
| mysqlx_ssl_crlpath                  |                 |
| mysqlx_ssl_key                      |                 |
| performance_schema_show_processlist | OFF             |
| ssl_ca                              | ca.pem          |
| ssl_capath                          |                 |
| ssl_cert                            | server-cert.pem |
| ssl_cipher                          |                 |
| ssl_crl                             |                 |
| ssl_crlpath                        |                 |
| ssl_fips_mode                      | OFF             |
| ssl_key                            | server-key.pem  |
+------------------------------------+-----------------+

> status
SSL:   Cipher in use is ECDHE-RSA-AES128-GCM-SHA256
```

11. View the contents of the ssl-client.xml file by running the following commands:

```
export SSL_CLIENT=/etc/hadoop/conf/ssl-client.xml
cat $SSL_CLIENT
```

12. Obtain the truststore's location and password by running the following commands:

```
export TRUSTSTORE_LOCATION=$(xmllint --xpath "//configuration/property[n
ame='ssl.client.truststore.location']/value/text()" $SSL_CLIENT)
```

```
export TRUSTSTORE_PASSWORD=$(xmllint --xpath "//configuration/property[n
ame='ssl.client.truststore.password']/value/text()" $SSL_CLIENT)
```

13. Verify the contents of the truststore by running the following command:

```
/usr/java/default/bin/keytool -list -rfc -keystore $TRUSTSTORE_LOCATION -
storetype JKS -storepass $TRUSTSTORE_PASSWORD
```

14. Import the MySQL root certificate by running the following command:

```
/usr/java/default/bin/keytool -importcert -alias mysql -file /var/lib/my
sql/ca.pem -keystore $TRUSTSTORE_LOCATION -storetype jks -noprompt -stor
epass $TRUSTSTORE_PASSWORD
```

15. Verify the contents of the truststore again by running the following command:

```
/usr/java/default/bin/keytool -list -rfc -keystore $TRUSTSTORE_LOCATION -
storetype JKS -storepass $TRUSTSTORE_PASSWORD
```

Results

When you install CDP Private Cloud Data Services after adding the MySQL root CA to the Cloudera Manager truststore, the installer propogates the MySQL root CA from the Cloudera Manager truststore to CDP Private Cloud. HMS in the default Database Catalog can now connect to the MySQL server on the base cluster using an SSL-encrypted connection.

Configuring MariaDB database to use SSL for Data Warehouse

SSL encrypts the connection between the MariaDB server and the Hive MetaStore (HMS) on the base cluster. You must enable SSL for the MariaDB database before setting up the CDP Private Cloud Data Services.

Procedure

1. SSH into the MariaDB database host.
2. Start the MariaDB server:

```
service mysqld start
```

3. Establish an encrypted connection with the client:

```
mysql -p --ssl=true
```

4. Verify whether SSL is enabled on MariaDB by running the following command:

```
mysql> show global variables like '%ssl%';
```

If SSL is enabled, you see the value of have_ssl equal to YES, as follows. Otherwise, you see the value of have _ssl equal to DISABLED:

```
+---------------+----------+
| Variable_name | Value    |
+---------------+----------+
| have_openssl  | YES      |
| have_ssl      | YES      |
| ...           | ...      |
```

If SSL is enabled, then skip to step 11.

5. Create a certificate authority by running the following commands:

```
mkdir /etc/my.cnf.d/ssl/
cd /etc/my.cnf.d/ssl/
openssl genrsa 2048 > ca-key.pem
```

6. Create a certificate for the server using the CA certificate generated earlier by running the following command:

```
openssl req -new -x509 -nodes -days 365000 -key ca-key.pem -out ca-cert.
pem
openssl req -newkey rsa:2048 -days 365 -nodes -keyout server-key.pem -out
 server-req.pem
openssl rsa -in server-key.pem -out server-key.pem
```

7. Create a certificate for the clients using the same CA certificate by running the following command:

```
openssl x509 -req -in server-req.pem -days 365 -CA ca-cert.pem -CAkey ca-
key.pem -set_serial 01 -out server-cert.pem
```

8. Add the following lines in the /etc/my.cnf.d/server.cnf file under the [mysqld] section:

```
ssl-ca=/etc/my.cnf.d/ssl/ca-cert.pem
ssl-cert=/etc/my.cnf.d/ssl/server-cert.pem
ssl-key=/etc/my.cnf.d/ssl/server-key.pem
bind-address=*
```

You can view the content of the server.cnf file by running the following command:

```
vim /etc/my.cnf.d/server.cnf
```

9. Restart the MariaDB server:

```
service mysqld restart
```

10. Check the SSL status by running the following commands:

```
mysql -p --ssl=true
> SHOW VARIABLES LIKE '%ssl%';
> status
```

Sample output:

```
> SHOW VARIABLES LIKE '%ssl%';
+---------------------+------------------------------------+
| Variable_name       | Value                              |
+---------------------+------------------------------------+
| have_openssl         | YES                                |
| have_ssl             | YES                                |
| ssl_ca               | /etc/my.cnf.d/ssl/ca-cert.pem      |
| ssl_capath           |                                    |
| ssl_cert             | /etc/my.cnf.d/ssl/server-cert.pem  |
| ssl_cipher           |                                    |
| ssl_crl              |                                    |
| ssl_crlpath          |                                    |
| ssl_key              | /etc/my.cnf.d/ssl/server-key.pem   |
| version_ssl_library  | OpenSSL 1.0.2k-fips  26 Jan 2017   |
+----------------------+------------------------------------+

> status
SSL:   Cipher in use is DHE-RSA-AES256-GCM-SHA384
```

11. View the contents of the ssl-client.xml file by running the following commands:

```
export SSL_CLIENT=/etc/hadoop/conf/ssl-client.xml
cat $SSL_CLIENT
```

12. Obtain the truststore's location and password by running the following commands:

```
export TRUSTSTORE_LOCATION=$(xmllint --xpath "//configuration/property[n
ame='ssl.client.truststore.location']/value/text()" $SSL_CLIENT)
```

```
export TRUSTSTORE_PASSWORD=$(xmllint --xpath "//configuration/property[n
ame='ssl.client.truststore.password']/value/text()" $SSL_CLIENT)
```

13. Verify the contents of the truststore by running the following command:

```
/usr/java/default/bin/keytool -list -rfc -keystore $TRUSTSTORE_LOCATION -
storetype JKS -storepass $TRUSTSTORE_PASSWORD
```

14. Import the MariaDB root certificate by running the following command:

```
/usr/java/default/bin/keytool -importcert -alias mariadb -file /etc/my.c
nf.d/ssl/ca-cert.pem -keystore $TRUSTSTORE_LOCATION -storetype jks -nopr
ompt -storepass $TRUSTSTORE_PASSWORD
```

15. Verify the contents of the truststore again by running the following command:

```
/usr/java/default/bin/keytool -list -rfc -keystore $TRUSTSTORE_LOCATION -
storetype JKS -storepass $TRUSTSTORE_PASSWORD
```

Results

When you install CDP Private Cloud Data Services after adding the MariaDB root CA to the Cloudera Manager truststore, the installer propogates the MariaDB root CA from the Cloudera Manager truststore to CDP Private Cloud. HMS in the default Database Catalog can now connect to the MariaDB server on the base cluster using an SSL-encrypted connection.

Configuring Oracle database to use SSL for Data Warehouse

You must enable SSL for the Oracle database before setting up the CDP Private Cloud Data Services. Enabling SSL establishes a secure channel between the client (CDP-side) and the server (Oracle database server).

About this task

To enable SSL, you need to configure SSL only on the server side. The client-side configurations are present in CDP.

Procedure

1. SSH into the Oracle database server host.
2. Change to the "oracle" user as follows:

   ```
   sudo -su oracle
   ```

3. Append the location of ORACLE_HOME to the PATH environment variable by running the following commands:

   ```
   export ORACLE_HOME=/opt/oracle/product/19c/dbhome_1
   export PATH=${PATH}:${ORACLE_HOME}/bin
   ```

4. Create an auto-login wallet by running the following command:

   ```
   orapki wallet create -wallet /opt/oracle/product/19c/dbhome_1/wallet -au
   to_login
   ```

   An auto-login wallet uses SSL's single sign-on functionality. The users do not need to specify password each time they open the wallet.
5. Add a self-signed certificate to this wallet by running the following command:

   ```
   orapki wallet add -wallet /opt/oracle/product/19c/dbhome_1/wallet -dn "C
   N=server" -keysize 4096 -self_signed -validity 365
   ```

6. Export the certificate from the Oracle wallet by running the following command:

   ```
   orapki wallet export -wallet /opt/oracle/product/19c/dbhome_1/wallet -dn
    "CN=server" -cert server_ca.cert
   ```

   This exports a certificate with the subject's distinguished name (-dn) (CN=server) from a wallet to the file that is specified by -cert (server_ca.cert).
7. Add the following lines to the /opt/oracle/product/19c/dbhome_1/network/admin/listener.ora configuration file:

   ```
   SSL_CLIENT_AUTHENTICATION = FALSE
   WALLET_LOCATION =
     (SOURCE =
       (METHOD = FILE)
       (METHOD_DATA =
         (DIRECTORY = /opt/oracle/product/19c/dbhome_1/wallet)
       )
     )
   Register a new address in LISTENER:
   (ADDRESS = (PROTOCOL = TCPS)(HOST = [***HOST***])(PORT = 2484))
   ```

8. Add the following lines to the /opt/oracle/product/19c/dbhome_1/network/admin/sqlnet.ora profile configuration file:

   ```
   SSL_CLIENT_AUTHENTICATION = FALSE
   WALLET_LOCATION =
     (SOURCE =
       (METHOD = FILE)
       (METHOD_DATA =
         (DIRECTORY = /opt/oracle/product/19c/dbhome_1/wallet)
       )
     )
   ```

9. Add the following lines to the /opt/oracle/product/19c/dbhome_1/network/admin/tnsnames.ora configuration file:

   ```
   ORCLPDB1_SSL =
   ```

```
     (DESCRIPTION =
       (ADDRESS = (PROTOCOL = TCPS)(HOST = [***HOST***])(PORT = 2484))
       (CONNECT_DATA =
         (SERVER = DEDICATED)
         (SERVICE_NAME = ORCLPDB1)
       )
       (SECURITY =
         (MY_WALLET_DIRECTORY = /opt/oracle/product/19c/dbhome_1/wallet)
       )
     )
```

10. Restart the listener by running the following commands:

```
lsnrctl stop
lsnrctl start
```

11. Copy the content of the certificate that you exported earlier and add it to the keystore on the base cluster instances.

    Paste the copied content to the ca-cert.pem file.
12. Fetch the keystore password from the /etc/hadoop/conf/ssl-client.xml file by running the following command:

```
/usr/java/default/bin/keytool -importcert -alias oracle -file ca-cert.pe
m -keystore /var/lib/cloudera-scm-agent/agent-cert/cm-auto-global_trusts
tore.jks -storetype jks -noprompt -storepass [***PASSWORD***]
```

13. Log in to Cloudera Manager as an Administrator.
14. Go to Clusters Hive service Configuration Hive Metastore Server Advanced Configuration Snippet (Safety Valve) for hive-site.xml and click + to add the following:
    - Name: javax.jdo.option.ConnectionURL
    - Value: jdbc:oracle:thin:@tcps://[***BASE_CLUSTER_HOSTNAME***]:2484/ORCLPDB1?javax.net.ssl.trustStore=/var/lib/cloudera-scm-agent/agent-cert/cm-auto-global_truststore.jks&javax.net.ssl.trustStorePassword=[***PASSWORD***]&oracle.net.ssl_server_dn_match=false
15. Change the port to 2484 in the Hive Metastore Database Port field.
16. Click Save Changes.
17. Restart the Hive service.

Disable the SSL or TLS requirement for HMS database

Cloudera Data Warehouse (CDW) on Private Cloud requires that you enable SSL or TLS on the database that you use for the Hive MetaStore (HMS) on the base cluster. However, if you have not configured SSL or TLS on the HMS database and you want to continue using CDW, then you must disable the SSL requirement, so that the Database Catalog does not fail to start in an attempt to establish a secure connection with an unsecured database.

About this task

The disable-sslmode.sh script used to disable CDW's requirement to have an SSL-enabled database is designed to work on Machintosh OS.

Before you begin

You must have the DWAdmin role and must have activated an environment in CDW.

Procedure

1. Log in to the Data Warehouse service as DWAdmin.

2. **2.** Obtain the namespace of the Database Catalog from the Database Catalog tile.

   The Database Catalog namespace is on the line under the Environment name.
3. **3.** Open a terminal session on your computer.
4. **4.** Install “yq” and “kubectl” command-line tools.
5. **5.** Create a work directory.

   You use this directory to create and save the disable-sslmode.sh file, which you create in the next step.
6. **6.** Create the disable-sslmode.sh file on your computer by copying the following lines:

```
#!/bin/bash
set -euo pipefail
#set -euvx


SHORT=k:,n:,c:,u,a,b:,w:,h

>&2 echo "OS-Type: ${OSTYPE}"

if [[ "${OSTYPE}" == "darwin"* ]]; then
  >&2 echo "Using only short options in getopt"
  OPTS=$(getopt $SHORT "$@")
#elif [[ "${OSTYPE}" == "linux-gnu"* ]]; then
elif [[ "${OSTYPE}" == "linux-"* ]]; then
  >&2 echo "Using both short and long options in getopt"
  LONG=kubeconfig:,namespace:,config-map:,clean-up,apply,backup:,working-d
irectory:,help
  OPTS=$(getopt -a -n generate --options $SHORT --longoptions $LONG -- "$
@")
else
  >&2 echo "Unsupported OS type: ${OSTYPE}"
  exit 249
fi

eval set -- "${OPTS}"

usage() {
  >&2 echo <<EOF "Updates the SSL mode of javax.jdo.option.ConnectionURL
 in hive-site.xml of the specified config map in specified namespace.

Please take attention that during the update the kube config file is cop
ied into a temporary directory in /tmp

Prerequisites:
    - kubectl >= 1.23
    - yq >= 4.x

Usage: $0 -k|--kubeconfig <path-to-kube-config-file> -n|--namespace <k8s-n
amespace> [-s|--ssl-mode <ssl-mode-to-be-set>] [-c|--config-map <name-of-
config-map-to-update>]
Mandatory parameters:
    -k|--kubeconfig <path-to-a-kubeconfig-yaml-file>
        the location of kube config file to use
    -n|--namespace <namespace-name>
        the kubernetes namespace to edit

Optional parameters:
    -c|--config-map <config-map-name-to-update>
        the name of config map to update
        default: hive-conf-metastore

    -w|--working-directory <a-directory>
        a directory where temporary created files are stored
        if not specified then something is created under /tmp using com
mand mktemp -d

    -u|--clean-up
        deletes the resources created during update

    -a|--apply
        applies the changes
        if not specified it only generates the new config map file but is
not applying that.

    -b|--backup <backup-file-name>
        creates a backup in tar.gz (aka tgz) format.
        it will contain all files in working-directory

    -h|--help
        this description
"
EOF
}

# KUBECONFIG="shared-os-dev-01.yaml"
# NAMESPACE="compute-sj-230509t1723-hive"
CONFIG_MAP_NAME="hive-conf-metastore"
# BACKUP_FILE_NAME="/dev/null"
APPLY="false"
CLEAN_UP="false"
echo ${OPTS}

while :
do
  case "$1" in
    -k | --kubeconfig )
      KUBECONFIG="$2"
      shift 2
      ;;
    -n | --namespace )
      NAMESPACE="$2"
      shift 2
      ;;
    -c | --config-map )
      CONFIG_MAP_NAME="$2"
      shift 2
      ;;
    -b | --backup )
      BACKUP_FILE_NAME="$2"
      >&2 echo "Creating backup at the end: ${BACKUP_FILE_NAME}"
      shift 2
      ;;
    -w | --working-directory )
      WD="$2"
      shift 2
      ;;
    -a | --apply )
      APPLY="true"
      >&2 echo "Applying changes at the end"
      shift 1
      ;;
    -u | --clean-up )
      CLEAN_UP="true"
      >&2 echo "Cleaning up at the end"
      shift 1
      ;;
    -h | --help )
```

```
      usage
      exit 2
      ;;
    --)
      shift
      break
      ;;
    *)
      >&2 echo "unexpected option: $1"
      ;;
  esac
done
which yq >/dev/null || (echo "yq not found"; exit 251)
YQ_CURRENT_VERSION=`yq --version | awk '{ print $4; }' | cut -c 2-`
YQ_REQUIRED_VERSION=4.18.1

if [ "$(printf '%s\n' "${YQ_REQUIRED_VERSION}" "${YQ_CURRENT_VERSION}" |
 sort -V | head -n1)" = "${YQ_REQUIRED_VERSION}" ]; then
  >&2 echo "version of yq (${YQ_CURRENT_VERSION}) is greater than or eq
ual to ${YQ_REQUIRED_VERSION} ... OK"
else
  >&2 echo "version of 'yq' must be greater than or equal to '${YQ_REQUIR
ED_VERSION}' but it is '${YQ_CURRENT_VERSION}' ... FAILED"
  exit 248
fi

which kubectl
which kubectl >/dev/null || (echo "kubectl not found"; exit 252)

if [[ -z "${KUBECONFIG+x}" ]]; then
  >&2 echo "ERROR: missing kubeconfig use  -k or --kubeconfig"
 usage
 exit 254
fi

if [[ ! -f ${KUBECONFIG} ]]; then
  >&2 echo "ERROR: kubeconfig file does not exist: ${KUBECONFIG}"
  exit 246
fi
KUBECTL_CLIENT_CURRENT_VERSION="$(kubectl --kubeconfig ${KUBECONFIG} ver
sion -o yaml | yq '.clientVersion | [.major,.minor] | join(".")')"
KUBECTL_CLIENT_REQUIRED_VERSION=1.23
if [ "$(printf '%s\n' "${KUBECTL_CLIENT_REQUIRED_VERSION}" "${KUBECTL_C
LIENT_CURRENT_VERSION}" | sort -V | head -n1)" = "${KUBECTL_CLIENT_REQUI
RED_VERSION}" ]; then
  >&2 echo "version of kubectl (${KUBECTL_CLIENT_CURRENT_VERSION}) is grea
ter than or equal to ${KUBECTL_CLIENT_REQUIRED_VERSION} ... OK"
else
  >&2 echo "version of 'kubectl' must be greater than or equal to '${KU
BECTL_CLIENT_REQUIRED_VERSION}' but it is '${KUBECTL_CLIENT_CURRENT_VERS
ION}' ... FAILED"
  exit 247
fi


if [[ -z "${NAMESPACE+x}" ]]; then
  >&2 echo "ERROR: missing namespace use  -n or --namespace"
 usage
 exit 253
fi

if [[ -z "${WD+x}" ]]; then
 WD=$(mktemp -d)
fi
```

```
if [[ -z ${BACKUP_FILE_NAME+x} ]]; then
 BACKUP_FILE_NAME="backup.${NAMESPACE}.${CONFIG_MAP_NAME}.$(date -u +%Y%m
%dT%H%M%SZ).tar.gz"
fi


[[ -d "${WD}" ]] || mkdir -p "${WD}"
cat <<EOF > "${WD}/logs"
Working directory: ${WD}
kubeconfig: ${KUBECONFIG}
namespace: ${NAMESPACE}
config-map: ${CONFIG_MAP_NAME}
apply: ${APPLY}
backup: ${BACKUP_FILE_NAME}
clean-up: ${CLEAN_UP}
EOF

>&2 cat "${WD}/logs"
LOCAL_KUBECONFIG="$(basename "${KUBECONFIG}")"
cp "${KUBECONFIG}" "${WD}/${LOCAL_KUBECONFIG}"

pushd "${WD}"

ORIGINAL_CM_NAME="original.configmap.${CONFIG_MAP_NAME}.yaml"
ORIGINAL_HIVE_SITE_XML="${ORIGINAL_CM_NAME}.hive-site.xml"
UPDATED_CM_NAME="updated.configmap.${CONFIG_MAP_NAME}.yaml"
export UPDATED_HIVE_SITE_XML="${UPDATED_CM_NAME}.hive-site.xml"
env | grep -i path
>&2 echo "kubectl --kubeconfig \"${LOCAL_KUBECONFIG}\" get configmap \"$
{CONFIG_MAP_NAME}\" -n \"${NAMESPACE}\" -o yaml > \"${ORIGINAL_CM_NAME}\""
kubectl --kubeconfig "${LOCAL_KUBECONFIG}" get configmap "${CONFIG_MAP_NA
ME}" -n "${NAMESPACE}" -o yaml > "${ORIGINAL_CM_NAME}"
yq '.data."hive-site.xml"' > "${ORIGINAL_HIVE_SITE_XML}" < "${ORIGINAL_CM
_NAME}"
disableSslModeInPostgreSql() {
 local CONNECTION_URL="${1//sslmode=[^&]*/sslmode=disable}"
 CONNECTION_URL="${CONNECTION_URL//sslrootcert=[^&]*/}"
 CONNECTION_URL="${CONNECTION_URL//&&&*/&}"
 if [[ "${CONNECTION_URL:0-1}" == "&" ]]; then
  CONNECTION_URL="${CONNECTION_URL::-1}"
 fi
  if [[ "${CONNECTION_URL:0-1}" == "?" ]]; then
   CONNECTION_URL="${CONNECTION_URL::-1}"
 fi
 echo "${CONNECTION_URL}"
}

disableSslModeInMySql() {
 local CONNECTION_URL="${1//sslMode=[^&]*/sslMode=PREFERRED}"
 CONNECTION_URL="${CONNECTION_URL//trustCertificateKeyStoreUrl=[^&]*/}"
 CONNECTION_URL="${CONNECTION_URL//trustCertificateKeyStorePassword=[^&]*
/}"
 CONNECTION_URL="${CONNECTION_URL//&&&*/&}"
 if [[ "${CONNECTION_URL:0-1}" == "&" ]]; then
  CONNECTION_URL="${CONNECTION_URL::-1}"
 fi
  if [[ "${CONNECTION_URL:0-1}" == "?" ]]; then
   CONNECTION_URL="${CONNECTION_URL::-1}"
 fi
 echo "${CONNECTION_URL}"
}

disableSslModeInOracle() {
  local CONNECTION_URL="${1//javax.net.ssl.trustStore=[^&]*/}"
  CONNECTION_URL="${CONNECTION_URL//@tcps:/@tcp:}"
 CONNECTION_URL="${CONNECTION_URL//javax.net.ssl.trustStoreType=[^&]*/}"
 CONNECTION_URL="${CONNECTION_URL//javax.net.ssl.trustStorePassword=[^&
]*/}"
 CONNECTION_URL="${CONNECTION_URL//javax.net.ssl.keyStore=[^&]*/}"
 CONNECTION_URL="${CONNECTION_URL//javax.net.ssl.keyStoreType=[^&]*/}"
 CONNECTION_URL="${CONNECTION_URL//javax.net.ssl.keyStorePassword=[^&]*/}"
 CONNECTION_URL="${CONNECTION_URL//oracle.net.ssl_cipher_suites=[^&]*/}"
 CONNECTION_URL="${CONNECTION_URL//oracle.net.ssl_server_dn_match=[^&]*/}"
 CONNECTION_URL="${CONNECTION_URL//oracle.net.authentication_services=[^&
]*/}"
 CONNECTION_URL="${CONNECTION_URL//&&&*/&}"
  if [[ "${CONNECTION_URL:0-1}" == "&" ]]; then
   CONNECTION_URL="${CONNECTION_URL::-1}"
 fi
  if [[ "${CONNECTION_URL:0-1}" == "?" ]]; then
   CONNECTION_URL="${CONNECTION_URL::-1}"
 fi
 echo "${CONNECTION_URL}"
}

ORIGINAL_CONNECTION_URL=$(yq -p=xml '.Configuration.property[]|select(.na
me=="javax.jdo.option.ConnectionURL")|.value' < "${ORIGINAL_HIVE_SITE_XM
L}");
if [[ -n ${ORIGINAL_CONNECTION_URL} ]]; then
 >&2 echo "javax.jso.option.ConnectioURL=${ORIGINAL_CONNECTION_URL}";
 echo "javax.jso.option.ConnectioURL=${ORIGINAL_CONNECTION_URL}" >> logs;

 IFS=':' read -a token <<< "${ORIGINAL_CONNECTION_URL}"
 DRIVER="${token[1]}"
 >&2 echo "JDBC-Driver: ${DRIVER}"
 echo "Driver: ${DRIVER}" >> logs

 case "${DRIVER}" in
  postgres | postgresql )
   export NEW_CONNECTION_URL=$(disableSslModeInPostgreSql "${ORIGINAL_CON
NECTION_URL}")
   ;;
  mysql | mariadb )
   export NEW_CONNECTION_URL=$(disableSslModeInMySql "${ORIGINAL_CONNEC
TION_URL}")
   ;;
  oracle )
   export NEW_CONNECTION_URL=$(disableSslModeInOracle "${ORIGINAL_CONNE
CTION_URL}")
   ;;
  * )
   >&2 echo "Unsupported JDBC-Driver: ${DRIVER}"
   exit 250
   ;;
 esac


 >&2 echo "New ConnectionURL: ${NEW_CONNECTION_URL}";
 echo "New ConnectionURL: ${NEW_CONNECTION_URL}" >> logs;
 yq -p=xml '.Configuration.property|=map(select(.name=="javax.jdo.option
.ConnectionURL").value=env(NEW_CONNECTION_URL))' -o=xml > "${UPDATED_HIV
E_SITE_XML}" 2>> logs < "${ORIGINAL_HIVE_SITE_XML}"
 kubectl --kubeconfig "${LOCAL_KUBECONFIG}" get configmap "${CONFIG_MAP
_NAME}" -n "${NAMESPACE}" -o yaml | yq '.data."hive-site.xml" |= load_st
r(env(UPDATED_HIVE_SITE_XML))' > "${UPDATED_CM_NAME}" 2>>logs

 if [[ ${APPLY} == "true" ]]; then
```

```
  >&2 echo "Applying: kubectl --kubeconfig \"${LOCAL_KUBECONFIG}\" apply -
f ${UPDATED_CM_NAME}"
  echo "Applying: kubectl --kubeconfig \"${LOCAL_KUBECONFIG}\" apply -f ${
UPDATED_CM_NAME}" >> logs
  kubectl --kubeconfig "${LOCAL_KUBECONFIG}" apply -f "${UPDATED_CM_NAME}"
 >> logs 2>&1
 fi

fi;

popd

if [[ -n ${BACKUP_FILE_NAME+x} ]]; then
 >&2 echo "Creating backup from working-directory: ${BACKUP_FILE_NAME}"
 echo "Creating backup from working-directory: ${BACKUP_FILE_NAME}" >> "${
WD}/logs"
 tar czf "${BACKUP_FILE_NAME}" -C "${WD}" .
fi

if [[ "${CLEAN_UP}" == "true" ]]; then
 >&2 echo "Clean-up working-directory: ${WD}"
 rm -rf "${WD}"
fi
```

7. Run the disable-sslmode.sh script as follows:

```
./disable-sslmode.sh -k [***KUBECONFIG-FILENAME***].yaml -n [***DBC-NAME
SPACE***] -c hive-conf-metastore -w [***WORK-DIRECTORY***]
```

Replace [***KUBECONFIG-FILENAME***] with the actual kubeconfig yaml file and [***DBC-NAMESPAC
E***] with the Database Catalog namespace that you noted earlier.

Note: On MacOS, use short option flags in the command, such as “-n” instead of “--namespace”.

Sample output:

```
Working directory: /tmp/tmp.vQnpYUrIq5
kubeconfig: [***KUBECONFIG-FILENAME***].yaml
namespace: [***DBC-NAMESPACE***]
config-map: hive-conf-metastore
apply: true
backup: backup.[***DBC-NAMESPACE***].hive-conf-metastore.20230517T13195
5Z.tar.gz
clean-up: false
.
.
.
JDBC-Driver: postgresql
New ConnectionURL: jdbc:postgresql://postgres-service-default-warehouse:
5432/hive1?createDatabaseIfNotExist=true&sslmode=disable
Applying: kubectl --kubeconfig "[***KUBECONFIG-FILENAME***]" apply -f u
pdated.configmap.hive-conf-metastore.yaml
```

8. Note the “New ConnectionURL” from the logs of the disable-sslmode.sh script.

9. If the Database Catalog does not start automatically after the script completes running, then restart the metastore-0 and metastore-1 pods as follows:

```
kubectl --kubeconfig [***KUBECONFIG-FILENAME***].yaml delete pod metasto
re-0 -n [***DBC-NAMESPACE***]
```

   Wait for the pods to restart and be in the running state.

10. Go to Database Catalog ⋮ Edit CONFIGURATIONS Metastore and select hive-site from the Configuration files drop-down menu.
11. Add the value of the New ConnectionURL parameter in the javax.jdo.option.ConnectionURL field.

    This ensures that a correct connection URL is delegated to the Virtual Warehouses from the Database Catalog.
12. Click Apply Changes.

SSL-enabled endpoints for Virtual Warehouse clients in CDW on Private Cloud

In Cloudera Data Warehouse (CDW) Private Cloud, all client endpoints have been SSL-enabled. This requires that you configure the SSL certificates for client endpoints.

The client endpoints for web applications and Virtual Warehouse client URLs are SSL-enabled. The following endpoints use the OpenShift/Embedded Container Service cluster default certificate:

- Hue
- Impala coordinator
- HiveServer2

Domain name changes

To use the OpenShift/Embedded Container Service cluster wildcard certificate, the DNS names have been changed. The environment ID sub domain from the domain name has been removed. This creates a flat DNA structure so the cluster wildcard certificate can be applied to the endpoints.

Generating a truststore for a self-signed certificate

You can query the service certificate and convert it to a JKS truststore using the following steps:

1. Retrieve the certificate:

```
$ openssl s_client -showcerts -connect hs2-my-cwh1.apps.cdw.mycloud.myfi
rm.com:443 -servername
hs2-my-cwh1.apps.cdw.mycloud.myfirm.com </dev/null|openssl x509 -outform
PEM > <mycertfile>.pem
```

2. Convert the PEM file to a truststore. You will be prompted for a password.

```
$ keytool -import -alias hs2-my-cw1.apps.cdw.mycloud.myfirm.com -file
 <mycertfile>.pem -keystore <mycert>.jks
```

Opening SSL-enabled connections with Database Catalog clients

The CDW Virtual Warehouse clients like beeline and impala-shell can open SSL-enabled connections as described in this section.

Beeline

A beeline connection can be created using a JDBC connection string. Specifying the username and password with the '-n' and the '-p' options returns an error. The beeline CLI prompts for credentials:

```
$ beeline
beeline> !connect
jdbc:hive2://hs2-my-cwh1.apps.cdw.mycloud.myfirm.com:443/default;transportMo
de=http;httpPath=cliservice;
     ssl=true;retries=3;sslTrustStore=<JKS-path>;trustStorePassword
=<***password***>
Enter username for jdbc:hive2://hs2-my-cwh1.apps.cdw.mycloud.myfirm.com:443/
default:<my-user-name
Enter password for jdbc:hive2://hs2-my-cwh1.apps.cdw.mycloud.myfirm.com:443/
default:<********>
```

Important: The value for <*JKS-path*> is generated in the above section "Generating a truststore for a self-signed certificate."

impala-shell

The impala-shell CLI opens a TLS/SSL-enabled connection when you use the `--ssl` option. If `--ca_cert` is not set, impala-shell enables TLS/SSL, but does not validate the server certificate. Set the `--ca_cert` CLI option to the local path name that points to the third-party CA certificate, or to a copy of the server certificate in the case you have a self-signed server certificate:

```
$ impala-shell --protocol='hs2-http' -i "coordinator-my-iwh2.apps.cdw.myclou
d.myfirm.com:443" --ssl
```

OpenShift routes

OpenShift routes are used to expose the user-facing services in the CDW Private Cloud deployment. Route objects can perform edge TLS termination using the cluster-deployed certificate for the endpoints. If the cluster certificate must be rotated, the routes can pick up the new certificate automatically. It is not necessary to re-deploy or to manually configure the service in order to pick up the changes.

HDFS encryption in the context of Data Warehouse on Private Cloud

Cloudera Data Warehouse (CDW) Data Service and its components such as Hive, Impala, and Hue can read and write encrypted data to HDFS on a Private Cloud Base cluster.

How HDFS encryption works with CDW

Encryption and decryption of the data happens in the HDFS client library. The client library is part of the client application such as, Hive, Impala, Spark, or any service that is reading or writing the data. To use this functionality encapsulated in the HDFS client library, the services must have access to the Hadoop Key Management Server (KMS) to retrieve the master key. KMS is a part of the Ranger service that runs in the base cluster. Cloudera recommends that you configure a secure cluster and then establish a secure channel between the encrypted HDFS cluster and the service using TLS.

All authorizations need an authenticated security principal – a user id, if it is a user, or a service account if it is a service.

SQL engines, such as Impala and Hive, and Hue as their front-end user interface need to authenticate the user connecting to them in order to authorize the user for various database-level operations, such as SELECT or INSERT, and to pass this user to the HDFS encryption ops to be authorized for those.

Understanding HDFS encryption zones in the context of CDW

Encryption Zone (EZ) is a directory in HDFS whose contents are automatically encrypted during write operations and decrypted during read operations. CDW can access data stored on the base cluster's HDFS, which can be set up with HDFS encryption.

You can configure the base cluster to have one or more HDFS encryption zones, a sub-directory encrypted with a particular master key. You can, then, store Hive and Impala tables in that sub-directory or you can store the entire Hive and Impala Warehouse in an encryption zone, encrypting the tables and metadata. CDW can then access the shared data from a Virtual Warehouse running in that extension.

Important: You can copy data between two encryption zones. Moving data between the encryption zones is neither possible nor recommended. SQL commands such as Impala's LOAD DATA INPATH statement copy the data instead of moving it when the data needs to cross encryption zone boundaries (reencrypting the data as necessary).

Conditions for enabling Impala to read and write encrypted data to HDFS

To access data in an encryption zone, you must set up authorization for various user principals.

To allow Impala to read and write encrypted data stored on HDFS, you must meet the following authorization permissions:

- You must have permissions to perform key operations for creating and accessing keys that encrypt your encryption zone (one key per zone).
- You must have read and write permissions to the HDFS sub-directory, which is your encryption zone.
- You must have permissions to perform various actions in the Hadoop SQL policy area in Ranger. For example, the ability to specify the LOCATION clause for a CREATE TABLE statement is a specific permission you have to grant to a user, which may be necessary if you have an encryption zone outside your warehouse directory and you want to write or read data there.

Encryption keys for the encryption zones are also managed in Ranger on the base cluster, together with their permissions (key ACLs).

You must grant these permissions to the user identities fetched from LDAP to make the base cluster and the CDW cluster refer to the same user identities in Ranger on the base cluster. You must configure the Ranger UserSync so that the base cluster can pull in the LDAP user identities.

You must also configure the Management Console to point to the same LDAP instance for authentication so that the base cluster and the CDW clusters are synchronized for user authentication.

If you are using Impala on encrypted source data, then ensure that data written to disk temporarily during processing is encrypted to keep the data confidential even during processing. For more information, see *Configuring Impala Virtual Warehouses to encrypt spilled data in CDW on Private Cloud*.

Note: The CDW cache is not encrypted. You must use a third-party disk encryption solution to encrypt data that is cached. Encrypting and decrypting data can degrade performance.

Related Information

HDFS Transparent Encryption (Cloudera documentation)
Transparent Encryption in HDFS (Apache documentation)
HDFS encryption: Rename and Trash considerations
HDFS encryption: Distcp considerations
Creating Encryption Zones
Transparent Encryption Recommendations for Impala
Transparent Encryption Recommendations for Hive
Transparent Encryption Recommendations for Hue
Configuring Impala Virtual Warehouses to encrypt spilled data in CDW on Private Cloud

Enabling browsing files on Kerberized HDFS from Hue in CDW Private Cloud

Because SPNEGO is disabled in Cloudera Data Warehouse (CDW) Private Cloud, by default, you cannot browse files on Kerberized HDFS cluster using the Hue File Browser. To enable access, you must set the security_enabled property to true in the Hue Advanced Configurations (hue-safety-valve) in CDW.

Procedure

1. Log in to the Data Warehouse service as DWAdmin.
2. Select a Virtual Warehouse on which you want to enable access to the Kerberized HDFS cluster from Hue and click ⋮ Edit .

   The **Virtual Warehouse Details** page is displayed.
3. Go to Configurations Hue , select hue-safety -valve from the Configuration files drop-down list, and add the following lines:

```
[hadoop]
[[hdfs_clusters]]
[[[default]]]
security_enabled=true
```

4. Click Apply Changes.

Configuring Impala Virtual Warehouses to encrypt spilled data in CDW on Private Cloud

If you have encrypted HDFS on the base CDP cluster, then Cloudera recommends that you configure an Impala Virtual Warehouse to write temporary data to disk during query processing in an encrypted format using the AES-256-CFB encryption for complete security.

About this task

In CDP Private Cloud, the temporary data is spilled to the local storage, the location of which is hard coded by the system.

⚠ **Important:** Impala does not selectively encrypt data based on whether the source data is already encrypted in HDFS. This results in at most 15 percent performance degradation when data is spilled.

Procedure

1. Log in to the Data Warehouse service as an administrator.
2. Go to Impala Virtual Warehouse ⋮ Edit CONFIGURATIONS Impala coordinator and select flagfile from the Configuration files drop-down list.
3. Set the value of the disk_spill_encryption property to true.
4. Click APPLY.
5. Go to the Impala executor tab and select flagfile from the Configuration files drop-down list.
6. Set the value of the disk_spill_encryption property to true.
7. Click APPLY.

8. Restart the Impala Virtual Warehouse.