Cloudera Runtime 7.2.18

Securing Apache Impala

Date published: 2020-11-30
Date modified: 2024-03-25

https://docs.cloudera.com/

Legal Notice

Contents

Securing Impala

You must be aware of this set of security features Impala provides to protect your critical and sensitive data.

The Impala security features have several objectives.

- At the basic level, security prevents accidents or mistakes that could disrupt application processing, delete or corrupt data, or reveal data to unauthorized users.
- Advanced security features and practices can harden the system against malicious users trying to gain unauthorized access or perform other disallowed operations.
- The auditing feature provides a way to confirm that no unauthorized access occurred, and detect whether any such attempts were made.

Based on the above objectives, Impala security features are divided into the following categories.

Authentication

How does Impala verify the identity of the user to confirm that they really are allowed to exercise the privileges assigned to that user? Impala relies on the Kerberos subsystem for authentication.

Authorization

Which users are allowed to access which resources, and what operations are they allowed to perform? Impala relies on the open source Ranger project for authorization. By default (when authorization is not enabled), Impala does all read and write operations with the privileges of the impala user, which is suitable for a development/test environment but not for a secure production environment. When authorization is enabled, Impala uses the OS user ID of the user who runs `impala-shell` or other client program, and associates various privileges with each user.

TLS/SSL Encryption

This feature encrypts TLS/SSL network encryption, between Impala and client programs, and between the Impala-related daemons running on different nodes in the cluster.

Auditing

What operations were attempted, and did they succeed or not? This feature provides a way to look back and diagnose whether attempts were made to perform unauthorized operations. You use this information to track down suspicious activity, and to see where changes are needed in authorization policies. The audit data produced by this feature is collected by the Cloudera Manager product and then presented in a user-friendly form by the Cloudera Manager product.

The following are the important guidelines to protect a cluster running Impala against accidents and mistakes, or malicious attackers trying to access sensitive data. See the subsequent topics that describe the security features in detail.

- Secure the root account. The root user can tamper with the `impalad` daemon, read and write the data files in HDFS, log into other user accounts, and access other system services that are beyond the control of Impala.
- Restrict membership in the sudoers list (in the /etc/sudoers file). The users who can run the sudo command can do many of the same things as the root user.
- Ensure the Hadoop ownership and permissions for Impala data files are restricted.

  The Impala authorization feature makes use of the HDFS file ownership and permissions mechanism.

  1. Set up users and assign them to groups at the OS level, corresponding to the different categories of users with different access levels for various databases, tables, and HDFS locations (URIs).
  2. Create the associated Linux users using the `useradd` command if necessary.
  3. Add Linux users to the appropriate groups with the `usermod` command.
- Ensure the Hadoop ownership and permissions for Impala log files are restricted.

  If you issue queries containing sensitive values in the WHERE clause, such as financial account numbers, those values are stored in Impala log files in the Linux file system, and you must secure those files.

- Ensure that the Impala web UI (available by default on port 25000 on each Impala node) is password-protected.
- Create a policy file that specifies which Impala privileges are available to users in particular Hadoop groups (which by default map to Linux OS groups). Create the associated Linux groups using the `groupadd` command if necessary.
- Design your databases, tables, and views with database and table structure to allow policy rules to specify simple, consistent rules.

  For example, if all tables related to an application are inside a single database, you can assign privileges for that database and use the * wildcard for the table name. If you are creating views with different privileges than the underlying base tables, you might put the views in a separate database so that you can use the * wildcard for the database containing the base tables, while specifying the specific names of the individual views.
- Enable authorization for all impalad daemons.

  The authorization feature does not apply to the `statestored` daemon, which has no access to schema objects or data files.
- Set up authentication using Kerberos, to make sure users really are who they say they are.

Related Information

Configuring Impala Web UI

Configuring Impala TLS/SSL

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

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

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

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

Using TLS/SSL with Business Intelligence Tools

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

Configuring TLS/SSL Communication for the Impala Shell

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

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

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

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

Specifying TLS/SSL Minimum Allowed Version and Ciphers

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

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

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

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

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

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

Impala Authentication

Authentication is the mechanism to ensure that only specified hosts and users can connect to Impala.

Authentication feature verifies that when clients connect to Impala, they are connected to a legitimate server. The feature prevents spoofing such as *impersonation* (setting up a phony client system with the same account and group names as a legitimate user) and *person-in-the-middle attacks* (intercepting application requests before they reach Impala and eavesdropping on sensitive information in the requests or the results).

Impala automatically handles both Kerberos and LDAP authentication. Each `impalad` daemon can accept both Kerberos and LDAP requests through the same port. No special actions need to be taken if some users authenticate through Kerberos and some through LDAP.

You can also make proxy connections to Impala through Apache Knox.

Regardless of the authentication mechanism used, Impala always creates HDFS directories and data files owned by the same user (typically impala). Once you are finished setting up authentication, set up authorization to control user-level access to databases, tables, columns, partitions when they connect through Impala.

Configuring Kerberos Authentication for Impala

Requirements for Using Impala with Kerberos

Important:

- If you plan to use Impala in your cluster, you must configure your KDC to allow tickets to be renewed, and you must configure krb5.conf to request renewable tickets. Typically, you can do this by adding the max_renewable_life setting to your realm in kdc.conf, and by adding the renew_lifetime parameter to the libdefaults section of krb5.conf.

Impala supports the Cloudera ODBC driver and the Kerberos interface provided. To use Kerberos through the ODBC driver, the host type must be set depending on the level of the ODBD driver:

- SecImpala for the ODBC 1.0 driver.
- SecBeeswax for the ODBC 1.2 driver.
- Blank for the ODBC 2.0 driver or higher, when connecting to a secure cluster.
- HS2NoSasl for the ODBC 2.0 driver or higher, when connecting to a non-secure cluster.

Enabling Kerberos in Impala-shell

To enable Kerberos in the Impala shell, start the `impala-shell` command using the -k flag.

Enabling Access to Internal Impala APIs for Kerberos Users

For applications that need direct access to Impala APIs, without going through the HiveServer2 or Beeswax interfaces, you can specify a list of Kerberos users who are allowed to call those APIs. By default, the impala and hdfs users are the only ones authorized for this kind of access. Any users not explicitly authorized through the internal _principals_whitelist configuration setting are blocked from accessing the APIs. This setting applies to all the Impala-related daemons, although currently it is primarily used for HDFS to control the behavior of the catalog server.

Mapping Kerberos Principals to Short Names for Impala

Impala can support the additional mapping rules that will be inserted before rules generated from the list of trusted realms and before the default rule. The support is disabled by default in Impala.

To enable mapping Kerberos principals to short names:

1. In Cloudera Manager, select the Impala service.
2. In the Configuration tab, select Impala (Service-Wide) in Scope and Advanced in Category.
3. Select the Use HDFS Rules to Map Kerberos Principals to Short Names field.
4. Click Save Changes, and restart the Impala service.

Configuring LDAP Authentication

Impala uses LDAP for authentication, verifying the credentials of each user who connects through `impala-shell`, Data Explorer, a Business Intelligence tool, JDBC or ODBC applications.

About this task

Authentication is the process of allowing only specified named users to access the server (in this case, the Impala server). This feature is crucial for any production deployment, to prevent misuse, tampering, or excessive load on the server.

Only the connections between clients and Impala can be authenticated by LDAP.

Procedure

To enable and configure LDAP:

1. In Cloudera Manager, select the Impala service.
2. In the Configuration tab, type ldap in the search box. The fields for LDAP configuration will be listed.

3. Set the following fields to enable LDAP.

   Enable LDAP Authentication (enable_ldap_auth)

   Enables LDAP-based authentication between the client and Impala.

   LDAP URL (ldap_uri)

   Sets the URI of the LDAP server to use. Typically, the URI is prefixed with ldap://. You can specify secure SSL-based LDAP transport by using the prefix ldaps://. The URI can optionally specify the port, for example: ldap://ldap_server.example.com:389 or ldaps://ldap_server.example.com:636. (389 and 636 are the default ports for non-SSL and SSL LDAP connections, respectively.)

4. Set the following fields to support custom bind strings.

   When Impala connects to LDAP, it issues a bind call to the LDAP server to authenticate as the connected user. Impala clients, including the Impala-shell, provide the short name of the user to Impala. This is necessary so that Impala can use role-based access, which uses short names.

   However, LDAP servers often require more complex, structured usernames for authentication. Impala supports three ways of transforming the short name (for example, 'henry') to a more complicated string. If necessary, specify one of the following configuration options when starting the `impalad` daemon.

   Active Directory Domain (--ldap_domain)

   Replaces the username with a string *USERNAME@LDAP_DOMAIN*.

   LDAP BaseDN (--ldap_baseDN)

   Replaces the username with a “distinguished name” (DN) of the form: uid=*USERID*,ldap_baseDN. (This is equivalent to a Hive option).

   LDAP Pattern (--ldap_bind_pattern)

   This is the most general option, and replaces the username with the string *LDAP_BIND_PATTERN* where all instances of the string #UID are replaced with *USERID*. For example, an ldap_bind_pa ttern of "user=#UID,OU=foo,CN=bar" with a username of henry will construct a bind name of "use r=henry,OU=foo,CN=bar".

   The above options are mutually exclusive, and Impala does not start if more than one of these options are specified.

5. Set the following fields for secure LDAP connections.

   To avoid sending credentials over the wire in cleartext, you must configure a secure connection between both the client and Impala, and between Impala and the LDAP server. The secure connection could use SSL or TLS.

   Secure LDAP connections through SSL:

   For SSL-enabled LDAP connections, specify a prefix of ldaps:// instead of ldap://. Also, the default port for SSL-enabled LDAP connections is 636 instead of 389.

   Secure LDAP connections through TLS:

   TLS, the successor to the SSL protocol, is supported by most modern LDAP servers. Unlike SSL connections, TLS connections can be made on the same server port as non-TLS connections. To secure all connections using TLS, specify the following flags as startup options to the `impalad` daemon:

   Enable LDAP TLS (--ldap_tls_

   Tells Impala to start a TLS connection to the LDAP server, and to fail authentication if it cannot be done.

   LDAP Server CA Certificate (--ldap_ca_certificate)

   Specifies the location of the certificate in standard .PEM format. Store this certificate on the local filesystem, in a location that only the impala user and other trusted users can read.

6. Click Save Changes and restart the Impala service.

Enabling LDAP in Cloudera Data Explorer (Hue)

To connect to Impala using LDAP authentication through Hue, enable the LDAP setting in Hue.

Procedure

1. Go to the Data Explorer (Hue) service.
2. Click the Configuration tab.
3. Select ScopeData Explorer (Hue) Server.
4. Select CategoryAdvanced.
5. Add the following properties to the Data Explorer (Hue) Server Advanced Configuration Snippet (Safety Valve) for hue_safety_valve_server.ini property

```
[impala]
auth_username=<LDAP username of Hue user to be authenticated>
auth_password=<LDAP password of Hue user to be authenticated>
```

6. Click Save Changes.

Enabling LDAP Authentication for impala-shell

To connect to Impala using LDAP authentication, you specify command-line options to the `impala-shell` command interpreter and enter the password when prompted.

-l

Enables LDAP authentication.

-u

Sets the user. Per Active Directory, the user is the short username, not the full LDAP distinguished name. If your LDAP settings include a search base, use the --ldap_bind_pattern on the `impalad` daemon to translate the short user name from `impala-shell` automatically to the fully qualified name.

`impala-shell` automatically prompts for the password.

Configuring JWT Authentication

Impala shell users can now provide a JWT instead of a username/password to authenticate to an Impala instance. When JWT authentication is used, the impala shell enforces the use of the hs2-http protocol since the JWT is sent via the "Authentication" HTTP header.

Procedure

To enable and configure JWT:

1. In Cloudera Manager, select the Impala service.
2. In the Configuration tab, type jwt in the search box. The fields for JWT configuration will be listed.

3. Set the following fields to enable JWT.

JWKS URL (jwks_url)

URL where the JSON Web Key Set (JWKS) can be downloaded for JWT verification. The default value is none, users will need to get the JWKS URL from their authentication provider's documentation.

Username JWT Custom Claim (jwt_custom_claim_username)

JWT claim that contains the username to use when authenticating with Impala. The default value is none, users will need to reference their authentication provider's documentation to determine which JWT payload claim contains the username.

JWT Token Authentication (jwt_token_auth)

Determines if JWT authentication is allowed without TLS being enabled on connections to the Impala daemon. The default value is false. Impala writes a warning message in the log file if it's set as true since it's not secure. Only set as true for test environments without sensitive data.

JWT Validate Signature (jwt_validate_signature)

Determines if the signatures on incoming JWTs are validated against the JWKS. If set as true, Impala verifies the signature in JWT with pre-installed public keys. The default value is true. Only set as true for test environments without sensitive data.

JWKS Pull Timeout (jwks_pulling_timeout_s)

The time in seconds to wait for the JWKS to be downloaded from the specified URL before timing out. This flag is used when jwks_url is specified. The default value is 10 seconds.

JWKS Update Frequency (jwks_update_frequency_s)

The time in seconds to wait between re-downloading the JWKS from the specified URL. This flag is used when jwks_url is specified. The default value is 60 seconds.

Verify JWKS Server Certificate (jwks_verify_server_certificate)

Specifies if the TLS certificate of the JWKS server is verified when retrieving the JWKS from the specified JWKS URL. This should only be set to false for development / testing. The default value is true.

Enabling JWT Authentication for impala-shell

To connect to Impala using JWT authentication, you specify the following command-line options to the impala-shell command interpreter and enter the password when prompted.

-j, --jwt

indicates that JWT authentication will be used

--jwt_cmd

shell command to run to retrieve the JWT to be used for authentication

Note: When using the JWT authentication, impala shell uses the hs2-https protocol by default.

Examples

Reading the JWT from a file named my_jwt.txt.

```
impala-shell -i impala_hostname:port -j --ssl --jwt_cmd "cat my_jwt.txt"
```

Note: To read the JWT from a file, you must have created a file named "my_jwt.txt" and stored the JWT without the quotes.

Reading the JWT from an environment variable named MY_JWT_ENV_VAR

```
impala-shell --ssl -i impala_hostname:port -j --jwt_cmd "echo ${MY_JWT_ENV_V
AR}"
```

Reading the JWT from the clipboard (macOS only)

```
impala-shell --ssl -i impala_hostname:port -j --jwt_cmd "pbpaste"
```

Configuring OAuth Authentication

Impala shell users can now use OAuth for authentication instead of a username and password. When OAuth authentication is used, the Impala shell enforces the use of the hs2-http protocol since OAuth is sent through the "Authentication" HTTP header.

Procedure

To enable and configure OAuth:

1. In Cloudera Manager, select the Impala service.
2. In the Configuration tab, search for 'Impala Daemon Command Line Argument Advanced Configuration Snippet (Safety Valve)' to add the OAuth configuration fields.
3. Set the following fields to enable OAuth.
   a) OAUTH URL: oauth_jwks_url

      This is the web address where Impala gets the security keys needed to verify OAuth authentication. By default, no URL is set. You'll find the correct URL in your authentication provider's documentation.

   b) Username OAuth Custom Claim: oauth_jwt_custom_claim_username

      This setting tells Impala which part of the OAuth token contains the short username for authentication. By default, no specific claim is set. Refer to your authentication provider's documentation to find out which claim holds the username.

   c) OAuth Token Authentication: oauth_token_auth (Default: false)

      When true, OAuth authentication is enabled

   d) OAuth Validate Signature: oauth_jwt_validate_signature (Default: true)

      This setting determines if Impala checks the digital signature of incoming OAuth tokens against the downloaded security keys. With default value Impala verifies the token's signature using pre-installed public keys.

   e) OAuth JWKS Pull Timeout: oauth_jwks_pulling_timeout_s (Default: 10 sec)

      This is the maximum time (in seconds) Impala will wait to download the security keys from the specified URL before giving up. This setting applies only when an oauth_jwks_url is provided.

   f) OAuth JWKS Update Frequency: oauth_jwks_update_frequency_s (Default: 60 seconds)

      This is how often (in seconds) Impala will re-download the security keys from the specified URL to ensure they are up-to-date. This setting also applies only when an oauth_jwks_url is provided.

   g) Verify OAuth JWKS Server Certificate oauth_jwks_verify_server_certificate (Default: true)

      This controls whether Impala verifies the security certificate of the server providing the OAuth security keys. For production environments, this should always be true. Only set it to false for development or testing purposes.
4. Click Save Changes and restart the Impala service.

Impala Authorization

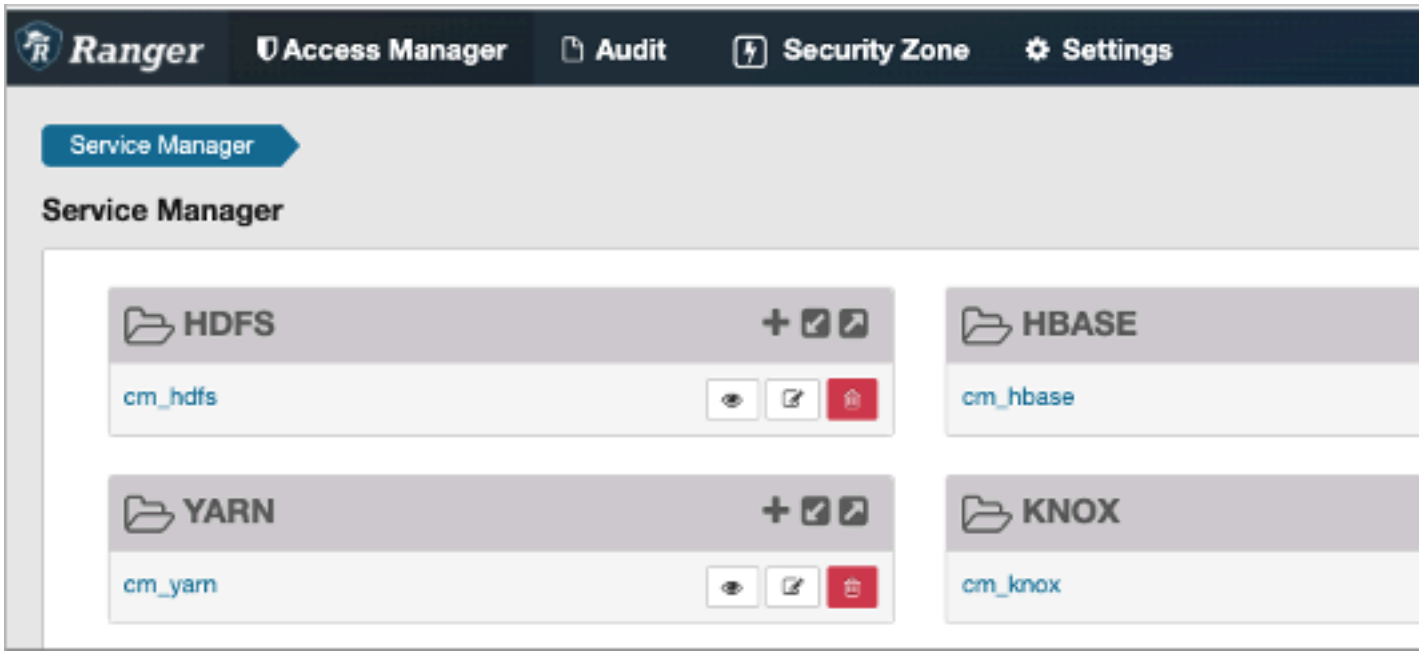
Authorization determines which users are allowed to access which resources, and what operations they are allowed to perform. You use Apache Ranger to enable and manage authorization in Impala.

Supported Ranger Features in Impala

- Resource-based and tag-based authorization policies
- Resource-based and tag-based column masking
- Row-level filtering is enabled by default

Using Resource-based Authorization Policies for Impala

In the Ranger Service Manager, you can use the Hadoop SQL preloaded resource-based and tag-based services and policies to authorize access to Impala:



You can configure services for Impala as described in Using Ranger to Provide Authorization in Cloudera.

For example, you can edit the all-global policy for an Impala user:

For information about using tag-based policies, see Tag-based column masking in Hive with Ranger policies.

Privilege Model

You set up privileges through the GRANT and REVOKE statements in either Impala or Hive. Then both components use those same privileges automatically.

By default, when authorization is not enabled, Impala does all read and write operations with the privileges of the impala user, which is suitable for a development/test environment but not for a secure production environment. When authorization is enabled, Impala uses the OS user ID of the user who runs `impala-shell` or other client programs, and associates various privileges with each user.

Privileges can be granted on different resources in the schema and are associated with a level in the resource hierarchy. A privilege on a particular resource automatically inherits the same privilege of its parent.

The resource hierarchy is:

```
Server
    URI
    Database
        Table
            Column
        Function
```

The table-level privileges apply to views as well. Anywhere you specify a table name, you can specify a view name instead.

You can specify privileges for individual columns.

The table below lists the minimum level of privileges and the scope required to run SQL statements in Impala. The following notations are used:

- The SERVER resource type in Ranger implies all databases, all tables, all columns, all UDFs, and all URLs.
    - **Note:** For Impala server level access, you must add the user to the below listed default policies in Ranger:

      all - database, table, column

      all - database, udf

      all - url
- ANY denotes the CREATE, ALTER, DROP, SELECT, INSERT, or REFRESH privilege.
- ALL privilege denotes the SELECT, INSERT, CREATE, ALTER, DROP, and REFRESH privileges.
- VIEW_METADATA privilege denotes the SELECT, INSERT, or REFRESH privileges.
- The parent levels of the specified scope are implicitly supported. For example, if a privilege is listed with the TABLE scope, the same privilege granted on DATABASE and SERVER will allow the user to run that specific SQL statement on TABLE.

Note: For an "impala" user to access any managed tables under the warehouse root directory or external tables with directories outside the warehouse root directory, the "impala" user must have HDFS RW access to the underlying paths that are referenced in a query.

Without this HDFS RW access, you may see Impala queries failing with HDFS errors such as this:

Error(13): Permission denied

Root cause: RemoteException: Permission denied: user=impala, access=WRITE, inode="/path/external/table":hive:supergroup:drwxr-xr-x

For example, to be able to run CREATE VIEW, you need the CREATE privilege on the database and the SELECT privilege on the source table.

SQL Statement	Privileges	Object Type / Resource Type
SELECT	SELECT	TABLE

WITH SELECT	SELECT	TABLE
EXPLAIN SELECT	SELECT	TABLE
INSERT	INSERT	TABLE
EXPLAIN INSERT	INSERT	TABLE
TRUNCATE	INSERT	TABLE
LOAD	INSERT	TABLE
	ALL	URI
CREATE DATABASE	CREATE	SERVER
CREATE DATABASE LOCATION	CREATE	SERVER
	ALL	URI
CREATE TABLE	CREATE	DATABASE
CREATE TABLE LIKE	CREATE	DATABASE
	VIEW_METADATA	TABLE
CREATE TABLE AS SELECT	CREATE	DATABASE
	INSERT	DATABASE
	SELECT	TABLE
EXPLAIN CREATE TABLE AS SELECT	CREATE	DATABASE
	INSERT	DATABASE
	SELECT	TABLE
CREATE TABLE LOCATION	CREATE	TABLE
	ALL	URI
CREATE VIEW	CREATE	DATABASE
	SELECT	TABLE
ALTER DATABASE SET OWNER	ALL WITH GRANT	DATABASE
ALTER TABLE	ALL	TABLE
ALTER TABLE SET LOCATION	ALL	TABLE
	ALL	URI
ALTER TABLE RENAME	CREATE	DATABASE
	ALL	TABLE
ALTER TABLE SET OWNER	ALL WITH GRANT	TABLE
ALTER VIEW	ALL	TABLE
	SELECT	TABLE
ALTER VIEW RENAME	CREATE	DATABASE
	ALL	TABLE
ALTER VIEW SET OWNER	ALL WITH GRANT	VIEW
DROP DATABASE	ALL	DATABASE
DROP TABLE	ALL	TABLE
DROP VIEW	ALL	TABLE
CREATE FUNCTION	CREATE	DATABASE
	ALL	URI

DROP FUNCTION	ALL	DATABASE
COMPUTE STATS	ALL	TABLE
DROP STATS	ALL	TABLE
INVALIDATE METADATA	REFRESH	SERVER
INVALIDATE METADATA <table>	REFRESH	TABLE
REFRESH <table>	REFRESH	TABLE
REFRESH AUTHORIZATION	REFRESH	SERVER
REFRESH FUNCTIONS	REFRESH	DATABASE
COMMENT ON DATABASE	ALL	DATABASE
COMMENT ON TABLE	ALL	TABLE
COMMENT ON VIEW	ALL	TABLE
COMMENT ON COLUMN	ALL	TABLE
DESCRIBE DATABASE	VIEW_METADATA	DATABASE
DESCRIBE <table/view>	VIEW_METADATA	TABLE
If the user has the SELECT privilege at the COLUMN level, only the columns the user has access will show.	SELECT	COLUMN
USE	ANY	TABLE
SHOW DATABASES	ANY	TABLE
SHOW TABLES	ANY	TABLE
SHOW FUNCTIONS	VIEW_METADATA	DATABASE
SHOW PARTITIONS	VIEW_METADATA	TABLE
SHOW TABLE STATS	VIEW_METADATA	TABLE
SHOW COLUMN STATS	VIEW_METADATA	TABLE
SHOW FILES	VIEW_METADATA	TABLE
SHOW CREATE TABLE	VIEW_METADATA	TABLE
SHOW CREATE VIEW	VIEW_METADATA	TABLE
SHOW CREATE FUNCTION	VIEW_METADATA	DATABASE
SHOW RANGE PARTITIONS (Kudu only)	VIEW_METADATA	TABLE
UPDATE (Kudu only)	ALL	TABLE
EXPLAIN UPDATE (Kudu only)	ALL	TABLE
UPSERT (Kudu only)	ALL	TABLE
WITH UPSERT (Kudu only)	ALL	TABLE
EXPLAIN UPSERT (Kudu only)	ALL	TABLE
DELETE	ALL	TABLE
EXPLAIN DELETE (Kudu only)	ALL	TABLE

The privileges not listed in the table above will be silently ignored by Impala.

Changing Privileges in Impala

Privileges are managed via the GRANT and REVOKE SQL statements that require the Ranger service enabled.

Privileges can be also managed in Ranger UI. Especially, for attribute-based access control, Ranger UI is required to manage authorization.

Impala authorization policies are listed in the Hive service section in Ranger UI.

REFRESH AUTHORIZATION is not required when you make the changes to privileges within Impala. The changes are automatically propagated.

Changing Privileges from Outside of Impala

If you make a change to privileges in Ranger from outside of Impala, e.g. adding a user, removing a user, modifying privileges, there are two options to propagate the change:

- Use the ranger.plugin.hive.policy.pollIntervalMs property to specify how often to do a Ranger refresh. The property is specified in ranger-impala-security.xml in the conf directory under your Impala home directory.
- Run the REFRESH AUTHORIZATION statement to force a refresh.

Warning: As INVALIDATE METADATA is an expensive operation, you should use it judiciously.

Granting Privileges on URI

URIs represent the file paths you specify as part of statements such as CREATE EXTERNAL TABLE and LOAD DATA. Typically, you specify what look like UNIX paths, but these locations can also be prefixed with hdfs:// to make clear that they are really URIs. To set privileges for a URI, specify the name of a directory, and the privilege applies to all the files in that directory.

URIs must start with hdfs://, s3a://, adl://, or file://. If a URI starts with an absolute path, the path will be appended to the default filesystem prefix. For example, if you specify:

```
GRANT ALL ON URI '/tmp';
```

The above statement effectively becomes the following where the default filesystem is HDFS.

```
GRANT ALL ON URI 'hdfs://localhost:20500/tmp';
```

When defining URIs for HDFS, you must also specify the NameNode. For example:

```
GRANT ALL ON URI file:///path/to/dir TO <principal>
          GRANT ALL ON URI hdfs://namenode:port/path/to/dir TO <principal>
```

Warning: Because the NameNode host and port must be specified, it is strongly recommended that you use High Availability (HA). This ensures that the URI will remain constant even if the NameNode changes. For example:

```
GRANT ALL ON URI hdfs://ha-nn-uri/path/to/dir TO <principal>
```

Note:

```
<principal> can be a user, or a group
```

Note:

- If a user uses Impala SQL engine to access the resource of the specified URL/URI provided as part of the SQL statement and as an admin if you must deny access permissions to this location for this particular user, you must add the permission "All" in the "HADOOP SQL" policy.
- However, if the user is using Hive as the execution engine and to deny access permission to a location, then you must add both the “Read” and “Write” permissions in the field of ‘Permissions’ for the corresponding deny conditions.
- If the same Ranger policy is shared by both Hive and Impala, then you must add “All”, “Read”, and “Write” to the field of ‘Permissions’ to enforce the policy.

Object Ownership

Object ownership for tables, views and databases is enabled by default in Impala.

To define owner specific privileges, go to Ranger UI and define appropriate policies on the {OWNER} user.

The CREATE statements implicitly make the user running the statement the owner of the object. For example, if *USER A* creates a database, *FOO*, via the CREATE DATABASE statement, *USER A* now owns the *FOO* database and is authorized to perform any operation on the *FOO* database.

An ownership can be transferred to another user or role via the ALTER DATABASE, ALTER TABLE, or ALTER VIEW with the SET OWNER clause.

Note: Currently, due to a known issue (IMPALA-8937), until the ownership information is fully loaded in the coordinator catalog cache, the owner of a table might not be able to see the table when executing the SHOW TABLES statement. The owner can still query the table.

Ranger Column Masking for Impala

Ranger column masking hides sensitive columnar data in Impala query output. For example, you can define a policy that reveals only the first or last four characters of column data. Column masking is enabled by default. To disable column masking, modify the following configuration property in all coordinators:

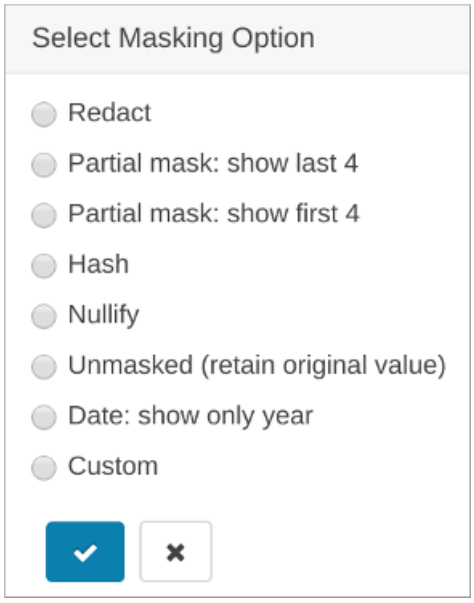
```
--enable_column_masking=false
```

This flag might be removed in a future release. The Impala behavior mimics Hive behavior with respect to column masking.

The following table lists all Impala-supported, built-in mask types for defining column masking in a policy using the Ranger REST API or Ranger UI:

Type	Name	Description	Transformer
MASK	Redact	Replace lowercase with 'x', uppercase with 'X', digits with '0'	mask({col})
MASK_SHOW_LAST_4	Partial mask: show last 4	Show last 4 characters; replace rest with 'x'	mask_show_last_n({col}, 4, 'x', 'x', 'x', -1, '1')
MASK_SHOW_FIRST_4	Partial mask: show first 4	Show first 4 characters; replace rest with 'x'	mask_show_first_n({col}, 4, 'x', 'x', 'x', -1, '1')
MASK_HASH	Hash	Hash the value	mask_hash({col})
MASK_NULL	Nullify	Replace with NULL	N/A
MASK_NONE	Unmasked (retain original value)	No masking	N/A
MASK_DATE_SHOW_YEAR	Date: show only year	Date: show only year	mask({col}, 'x', 'x', 'x', -1, '1', 1, 0, -1)
CUSTOM	Custom	Custom	N/A

The table includes the mask name as it appears in the Ranger UI.



Ranger Column Masking Limitations in Impala

- Column masking introduces unused columns during the query analysis and, consequently, additional SELECT privileges checks on all columns of the masked table.
- Impala might produce more than one audit log entry for a column involved in a column masking policy under all of these conditions: the column appears in multiple places in a query, the query is rewritten, or the query is re-analyzed.
- Column masking policies, shared between Hive and Impala, might be affected by SQL/UDF differences between Hive and Impala, as shown in the following example.

  For instance, UTF-8 strings containing non-ASCII characters are not guaranteed to work properly. Suppose a column masking policy masks the last two characters of a string: s => mask_last_n(s, 2, 'x', 'x', 'x', 'x'). Applying this policy, Hive properly masks SQL## to SQLxx, but the Impala masking produces SQL##xx because each Chinese character is encoded in 3 bytes. Impala and Hive do not handle different lengths in the same way.

Limitations on Mask Functions

The mask functions in Hive are implemented through GenericUDFs. Even though Impala users can call Hive UDFs, Impala does not yet support Hive GenericUDFs, so you cannot use Hive's mask functions in Impala. However, Impala has builtin mask functions that are implemented through overloads. In Impala, when using mask functions, not all parameter combinations are supported. These mask functions are introduced in Impala 3.4

The following list includes all the implemented overloads.

- Overloads used by Ranger default masking policies,
- Overloads with simple arguments,
- Overload with all arguments in int type for full functionality. Char argument needs to be converted to their ASCII value.

To list the available overloads, use the following query:

```
show functions in _impala_builtins like "mask*";
```

Note: An error message that states "No matching function with signature: mask..." implies that Impala does not contain the corresponding overload.

Related Information

GRANT statement

REVOKE statement
Resource-based column masking in Hive with Ranger policies
Tag-based column masking in Hive with Ranger policies
Apache Ranger documentation

Configuring Authorization

Enable Ranger authorization in Impala.

Procedure

1. In Cloudera Manager, navigate to the Impala service.
2. In the Configuration tab, type ranger in the search field.
3. Specify the values in the following fields.
   - Ranger Service: Select the Ranger service you are using. This is the server name to use when granting server level privileges.
   - Ranger Policy Cache Directory: Specify the directory where Ranger security policies are cached locally.
   - Ranger DFS Audit Path: Specify the DFS path on which Ranger audits are written. The special placeholder '${ranger_base_audit_url}' should be used as the prefix, in order to use the centralized location defined in the Ranger service.
   - Ranger Audit DFS Spool Dir: Specify the spool directory for Ranger audits being written to DFS.
   - Ranger Audit Solr Spool Dir: Specify the spool directory for Ranger audits being written to Solr.
4. Restart Impala and Hive.

Row-level filtering in Impala with Ranger policies

Row-level filtering policies are similar to other Ranger access policies. Apache Ranger row-level filtering policy allows to set access policies for rows when reading from a table.

You can use Apache Ranger row-level filtering policies to set access policies for rows when reading from a table. You can set filters for specific users, groups, and conditions. This release adds a new feature flag enable_row_filte ring which is set to be true by default. To enable row-filtering feature you must have set the column masking flag enable_column_masking to true since the row-level filtering depends on the column masking implementation. You can use this flag enable_row_filtering to disable this feature as required.

Note: Note that row filtering policies apply prior to any column masking policies, because column masking policies apply only on result data of the target table/view.

The following limitations apply when using row-level filters:

- Row filtering policies on nested tables can't be applied when nested collection columns(e.g. array, map columns) are used directly in the FROM clause. Such queries are currently forbidden.

  This is an example of the currently supported version of a query on the table my_tbl (id int, int_array array<int>) that has a row filter "id = 0".

  ```
  select a.item from my_tbl t, t.int_array a
  ```

  However an equivalent query below is not currently supported since it uses the int_array column non-relatively.

  ```
  select item from my_tbl.int_array
  ```

  Such queries are forbidden and you must rewrite them to the supported format until further notice.

- For information on the steps to set the row-level filtering using Apache Ranger, see the link provided under Related Information.

  Note: Hive features that are not supported by Impala should not be used in the filter since Impala can't evaluate them.

AES encryption and decryption support

Learn about the Impala support for Advanced Encryption Standard (AES) cryptography functions, including the various supported modes, key sizes, and function behavior.

Advanced Encryption Standard (AES) is a widely-used and secure way to protect sensitive data. It works by using a secret key to scramble your data in plaintext into a form that is unreadable as ciphertext. The Impala AES implementation supports two key sizes, 128-bit and 256-bit, to meet different security requirements.

Key functions and supported modes

Impala provides aes_encrypt() and aes_decrypt() functions as entry points for encryption and decryption operations, handling the processes based on user-provided keys, AES modes, and initialization vectors (IVs). The implementation includes key length and initialization vectors vector size validation to ensure data integrity and confidentiality.

These functions use a key, an encryption mode, and a special initialization vector (IV) to keep your data secure. Impala supports the following modes to provide you flexibility:

- Encryption Modes:
  - AES-GCM
  - AES-CTR
  - AES-CFB
- Decryption Modes:
  - AES-GCM
  - AES-ECB
  - AES-CTR
  - AES-CFB

The following table provides information about the available modes:

Mode	Description
AES-GCM (Galois/Counter Mode)	This mode combines the AES block cipher with the Galois/ Counter Mode (GCM) for authenticated encryption. It provides both confidentiality and integrity, making it suitable for secure communication and storage. Due to its strong security properties and efficiency, it is the default mode.
AES-ECB (Electronic Codebook)	This mode is a basic mode of operation in which each block of plaintext is encrypted independently with the same key. It is included for compatibility with legacy systems and is only supported for decryption.
AES-CTR (Counter Mode)	This mode turns a block cipher into a stream cipher by encrypting a counter value to produce a key stream. This key stream is then XORed with the plaintext to create the ciphertext, allowing for parallel encryption and decryption.
AES-CFB (Cipher Feedback Mode)	This mode uses ciphertext feedback to create a key stream. It operates on a block-by-block basis, in which the previous ciphertext block is encrypted and then XORed with the plaintext to produce the next ciphertext block.

Compatibility and error handling

For compatibility, AES-CTR and AES-CFB are provided as fallbacks for environments for which AES-GCM might not be supported, for example, OpenSSL versions lower than 1.0.1.

If a user-provided mode is a valid member of the AES_CIPHER_MODE modes but is not supported by the internal OpenSSL library, the library default mode is chosen.

If a user provides an unsupported or invalid mode, the system returns an error.

Encryption and decryption examples

Learn about examples for encrypting and decrypting data using various AES modes and demonstrates common error scenarios.

Basic encryption and decryption

This is the most common use case. You encrypt a string and then decrypt it to get the original value.

```
aes_encrypt(string, key, mode, iv) aes_decrypt(binary, key, mode, iv)
```

Encrypting a string with AES_128_GCM

Use this query to encrypt a string. The base64encode function converts the binary encrypted output into a text string that is easy to store and display.

```
SELECT base64encode(aes_encrypt('ABC',
 '1234567890123456','AES_128_GCM','1234567890123456'));
```

This example uses the following elements:

- 'ABC': The string to encrypt.
- '1234567890123456': The 128-bit encryption key.
- 'AES_128_GCM': The encryption mode.
- '1234567890123456': The initialization vector (IV).

Decrypting the encrypted a string

To reverse the process, use the base64decode function to convert the text back to binary, and then use the aes_decrypt function with the same key, mode, and IV to restore the original ABC string.

```
SELECT aes_decrypt(base64decode('x+am+BIqtrEK9FpC/
zrvpOycjQ=='),'1234567890123456','AES_128_GCM','1234567890123456');
```

Using different modes

The following examples showcase different AES modes, each with their own characteristics:

- The AES_128_ECB mode is the basic mode of operation.

  Note: This exampl shows that the AES_128_ECB mode is only supported for decryption, as an error occurs when the mode is used for encryption.

  - **Decrypt with ECB mode**

    ```
    select aes_decrypt(base64decode('y6Ss
    +zCYObpCbgfWfyNWTw=='),'1234567890123456','AES_128_ECB','');
    -- RESULTS
    ```

```
'ABC'
```

- The AES_256_GCM mode requires a longer 256-bit key.

  Note: The 12345678901234567890123456789012 key string is longer, matching the 256-bit requirement.

  - **Encrypt with AES_256_GCM**

    ```
    select base64encode(aes_encrypt('ABC',
     '12345678901234567890123456789012','AES_256_GCM','1234567890123456'));
    ```

  - **Decrypt with AES_256_GCM**

    ```
    select aes_decrypt(base64decode('F/
    DLkSwEikFOlqzXVCysy1JX7Q=='),'12345678901234567890123456789012','AES_256_GCM','1234
    ```

Common error scenarios

The following examples highlight important constraints and errors:

NULL Values – This example shows that the key is NULL, which will result in an error:

```
-- This will fail because the key is NULL
 select base64encode(aes_encrypt('ABC',NULL,'AES_256_GCM','123456789012345
6'));
```

Invalid Mode – This example shows using an incorrect mode string that will result in an error:

```
-- This will fail due to an invalid mode name
select base64encode(aes_encrypt('ABC', '12345678901234567890123456789012',
'AES_256_CTB','1234567890123456'));
```

Incorrect Key or IV Length – This example shows that the key must be either 128 or 256 bits long and the IV must also adhere to specific length requirements, otherwise an error will occur:

```
-- This will fail due to an incorrect key length
select base64encode(aes_encrypt('ABC', '123456789012345678901234567890121',
'AES_256_GCM','1234567890123456'));
```

Case Insensitivity – This example shows that the mode string is not case-sensitive.

```
-- This works, even with lowercase mode
select base64encode(aes_encrypt('ABC', '12345678901234567890123456789012','a
es_256_gcm','1234567890123456'));
```