

Managing Apache Kudu Security

Date published: 2020-11-04

Date modified: 2021-06-08



Legal Notice

© Cloudera Inc. 2025. 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

Security considerations.....	4
Security limitations.....	4
Authentication.....	4
Kudu authentication with Kerberos.....	4
Internal private key infrastructure (PKI).....	5
Authentication tokens.....	5
Client authentication to secure Kudu clusters.....	5
Coarse-grained authorization.....	5
Fine-grained authorization.....	6
Apache Ranger integration.....	6
Authorization tokens.....	7
Trusted users.....	8
Configure Kudu's integration with Apache Ranger.....	8
Ranger client caching.....	10
Encryption.....	11
Redaction.....	12
Configure a secure Kudu cluster using Cloudera Manager.....	12
Enable Kerberos authentication and RPC encryption.....	12
Configure coarse-grained authorization with ACLs.....	13
Enable Ranger authorization.....	13
Configure HTTPS encryption.....	14
Configure a secure Kudu cluster using flag safety valves.....	14

Security considerations

Kudu includes security features that allow Kudu clusters to be hardened against access from unauthorized users.

Kudu 1.3 (and higher) includes security features that allow Kudu clusters to be hardened against access from unauthorized users. Kudu uses strong authentication with Kerberos, while communication between Kudu clients and servers can now be encrypted with TLS. Kudu also allows you to use HTTPS encryption to connect to the web UI.

These security features should work seamlessly in Impala as well, as long as Impala's user is given permission to access Kudu.

Security limitations

Here are some limitations related to data encryption and authorization in Kudu.

- Data encryption at rest is not directly built into Kudu. Encryption of Kudu data at rest can be achieved through the use of local block device encryption software such as dmccrypt.
- Row-level authorization is not available.
- Kudu does not support configuring a custom service principal for Kudu processes. The principal must follow the pattern `kudu/<HOST>@<DEFAULT.REALM>`.
- Server certificates generated by Kudu IPKI are incompatible with bouncycastle version 1.52 and earlier.
- The highest supported version of the TLS protocol is TLSv1.2
- When you are creating a new Kudu service using the Ranger web UI, the Test Connection button is displayed. However, the TestConnection tab is not implemented in the Kudu Ranger plugin. As a result if you try to use it with Kudu it will fail, but that does not mean that the service is not working.

Authentication

Configure Kudu to enforce secure authentication among servers, and between clients and servers.

Authentication prevents untrusted actors from gaining access to Kudu, and securely identifies connecting users or services for authorization checks. Authentication in Kudu is designed to interoperate with other secure Hadoop components by utilizing Kerberos.

Kudu authentication with Kerberos

Configure authentication on Kudu servers. Authentication in Kudu is designed to interoperate with other secure Hadoop components by utilizing Kerberos.

Configure authentication on Kudu servers using the `--rpc_authentication` flag, which can be set to one of the following options:

- `required` - Kudu will reject connections from clients and servers who lack authentication credentials.
- `optional` - Kudu will attempt to use strong authentication, but will allow unauthenticated connections.
- `disabled` - Kudu will only allow unauthenticated connections.

By default, the flag is set to `optional`. To secure your cluster, set `--rpc_authentication` to `required`.

Scalability

Kudu authentication is designed to scale to thousands of nodes, which means it must avoid unnecessary coordination with a central authentication authority (such as the Kerberos KDC) for each connection.

Instead, Kudu servers and clients use Kerberos to establish initial trust with the Kudu master, and then use alternate credentials for subsequent connections. As described previously, the Kudu master issues internal X.509 certificates to tablet servers on startup, and temporary authentication tokens to clients on first contact.

Internal private key infrastructure (PKI)

Kudu uses an internal PKI to issue X.509 certificates to servers in the cluster. Connections between peers who have both obtained certificates will use TLS for authentication. In such cases, neither peer needs to contact the Kerberos KDC.

X.509 certificates are only used for internal communication among Kudu servers, and between Kudu clients and servers. These certificates are never presented in a public facing protocol. By using internally-issued certificates, Kudu offers strong authentication which scales to huge clusters, and allows TLS encryption to be used without requiring you to manually deploy certificates on every node.

Authentication tokens

After authenticating to a secure cluster, the Kudu client will automatically request an authentication token from the Kudu master. An authentication token encapsulates the identity of the authenticated user and carries the Kudu master's RSA signature so that its authenticity can be verified. This token will be used to authenticate subsequent connections.

By default, authentication tokens are only valid for seven days, so that even if a token were compromised, it cannot be used indefinitely. For the most part, authentication tokens should be completely transparent to users. By using authentication tokens, Kudu is able to take advantage of strong authentication, without paying the scalability cost of communicating with a central authority for every connection.

When used with distributed compute frameworks such as Apache Spark, authentication tokens can simplify configuration and improve security. For example, the Kudu Spark connector will automatically retrieve an authentication token during the planning stage, and distribute the token to tasks. This allows Spark to work against a secure Kudu cluster where only the planner node has Kerberos credentials.

Client authentication to secure Kudu clusters

Users running client Kudu applications must first run the `kinit` command to obtain a Kerberos ticket-granting ticket. For example:

```
kinit admin@EXAMPLE-REALM.COM
```

Once authenticated, you use the same client code to read from and write to Kudu servers with and without the Kerberos configuration.

Coarse-grained authorization

Kudu supports coarse-grained authorization checks for client requests based on the client's authenticated Kerberos principal (user or service). Access levels are granted based on whitelist-style Access Control Lists (ACLs), one for each level. Each ACL specifies a comma-separated list of users, or may be set to `*` to indicate that all authenticated users have access rights at the specified level.

The two levels of access which can be configured are:

- **Superuser** - Principals authorized as a superuser can perform certain administrative functions such as using the kudu command line tool to diagnose and repair cluster issues.

- User - Principals authorized as a user are able to access and modify all data in the Kudu cluster. This includes the ability to create, drop, and alter tables, as well as read, insert, update, and delete data. The default value for the User ACL is '*', which allows all users access to the cluster. However, if authentication is enabled, this will restrict access to only those users who are able to successfully authenticate using Kerberos. Unauthenticated users on the same network as the Kudu servers will be unable to access the cluster.



Note: Internally, Kudu has a third access level for the daemons themselves called Service. This is used to ensure that users cannot connect to the cluster and pose as tablet servers.

Fine-grained authorization

Kudu can be configured to enforce fine-grained authorization across servers. This ensures that users can see only the data they are explicitly authorized to see. Kudu supports this by leveraging policies defined in Apache Sentry 2.2 and later. Starting with Kudu 1.12.0, Kudu now supports fine-grained authorization by leveraging policies defined in Apache Ranger 2.1 and later.



Note: Since support for Apache Sentry authorization has been deprecated since Kudu 1.12.0 and may be completely removed in the future, fine-grained authorization via Apache Ranger is preferred going forward.



Note: Fine-grained authorization policies are not enforced when accessing the web UI. User data may appear on various pages of the web UI (e.g. in logs, metrics, scans, etc.). As such, it is recommended to either limit access to the web UI ports, or redact or disable the web UI entirely, as desired.

Apache Ranger integration

Apache Ranger models tabular objects are stored in a Kudu cluster in the following hierarchy: Database, Table, Column.



Note: Ranger allows you to add separate service repositories to manage privileges for different Kudu clusters. Depending on the value of the `ranger.plugin.kudu.service.name` configuration in the Ranger client, Kudu knows which service repository to connect to. For more details about Ranger service repository, see the [Apache Ranger documentation](#).

Database: Kudu does not have the concept of a database. Therefore, a database is indicated as a prefix of table names with the format `<database>.<table>`. Since Kudu's only restriction on table names is that they be valid UTF-8 encoded strings, Kudu considers special characters to be valid parts of database or table names. For example, if a managed Kudu table created from Impala is named `impala::bar.foo`, its database will be `impala::bar`.

Table: Is a single Kudu table.

Column: Is a column within a Kudu table.

In Ranger, privileges are also associated with specific actions. Access to Kudu tables may rely on privileges on the following actions:

- ALTER
- CREATE
- DELETE
- DROP
- INSERT
- UPDATE
- SELECT

There are two additional access types:

- ALL
- METADATA

If a user has the ALL privilege on a resource, they implicitly have privileges to perform any action on that resource that does not require the users to be a delegated admin.

If a user is granted any privilege, they are able to perform actions requiring METADATA (for example, opening a table) without having to explicitly grant them METADATA privileges.

Ranger supports a delegate admin flag which is independent of the action type. It is not implied by ALL and does not imply METADATA. This is similar to the GRANT OPTION part of the ALL WITH GRANT OPTION in SQL as it is required to modify privileges in Ranger and change the owner of a Kudu table.



Warning: A user with delegate admin privilege on a resource can grant any privilege to themselves and others.

While the action types are hierarchical, in terms of privilege evaluation, Ranger does not have the concept of hierarchy. For instance, if a user has SELECT privilege on a database, it does not imply that the user has SELECT privilege on every table belonging to that database.

However, Ranger supports privilege wildcard matching. For example, `db=a->table=\*` matches all the tables that belong to database a. Therefore, in Ranger users actually need the SELECT privilege granted on `db=a->table=*>column=*` to allow SELECT on every table and every column in database a.

Nevertheless, with Ranger integration, when a Kudu master receives a request, it consults Ranger to determine what privileges a user has. And the required policies documented in the `<<security.adoc#policy-for-kudu-masters, policy section>>` are enforced to determine whether the user is authorized to perform the requested action or not.



Note: Even though Kudu table names remain case sensitive with Ranger integration, policies authorization is considered case-insensitive.

In addition to granting privileges to a user by username, privileges can also be granted to table owners using the special {OWNER} username. These policies are evaluated only when a user tries to perform an action on a table that they own. For example, a policy can be defined for the {OWNER} user and `db=*>table=*` resource, and it will automatically be applied when any table is accessed by its owner. This way administrators do not need to choose between creating policies one by one for each table, and granting access to a wide range of users.



Warning: If a user has ALL and delegate admin privileges on a table only by ownership and no privileges by username, they can effectively lock themselves out by giving away the ownership.

Related Information

[Ranger User Guide](#)

Authorization tokens

Rather than having every tablet server communicate directly with the underlying authorization service (Ranger), privileges are propagated and checked via authorization tokens. These tokens encapsulate what privileges a user has on a given table. Tokens are generated by the master and returned to Kudu clients upon opening a Kudu table. Kudu clients automatically attach authorization tokens when sending requests to tablet servers.

Authorization tokens are a means to limiting the number of nodes directly accessing the authorization service to retrieve privileges. As such, since the expected number of tablet servers in a cluster is much higher than the number of Kudu masters, they are only used to authorize requests sent to tablet servers. Kudu masters fetch privileges directly from the authorization service or cache.

Similar to the validity interval for authentication tokens, to limit the window of potential unwanted access if a token becomes compromised, authorization tokens are valid for five minutes by default. The acquisition and renewal of a token is hidden from the user, as Kudu clients automatically retrieve new tokens when existing tokens expire.

When a tablet server that has been configured to enforce fine-grained access control receives a request, it checks the privileges in the attached token, rejecting it if the privileges are not sufficient to perform the requested operation, or if it is invalid (e.g. expired).

Trusted users

It may be desirable to allow certain users to view and modify any data stored in Kudu. Such users can be specified via the `--trusted_user_acl` master configuration. Trusted users can perform any operation that would otherwise require fine-grained privileges, without Kudu consulting the authorization service.

Additionally, some services that interact with Kudu may authorize requests on behalf of their end users. For example, Apache Impala authorizes queries on behalf of its users, and sends requests to Kudu as the Impala service user, commonly "impala". Since Impala authorizes requests on its own, to avoid extraneous communication between the authorization service and Kudu, the Impala service user should be listed as a trusted user.



Note: When accessing Kudu through Impala, Impala enforces its own fine-grained authorization policy. This policy is similar to Kudu's and can be found in the [Impala authorization documentation](#).

Configure Kudu's integration with Apache Ranger

Apache Ranger has wider adoption and provides a more comprehensive security features (such as attribute based access control, audit, etc) than Sentry. This topic provides information to configure Kudu with Apache Ranger.

About this task

For information about how to enable Ranger authorization using Cloudera Manager, see *Enable Ranger authorization*.



Note:

- Ranger is often configured with Kerberos authentication.
- Sentry integration can not be enabled at the same time with Ranger integration.

Procedure

1. After building Kudu from source, find the `kudu-subprocess.jar` under the build directory, for example `build/release/bin`.
Note its path, as it is the one to the JAR file containing the Ranger subprocess, which houses the Ranger client that Kudu will use to communicate with the Ranger server.
2. Use the `kudu table list` tool to find any table names in the cluster that are not Ranger-compatible, which are names that begin or end with a period (.). Also check that there are no two table names that only differ by case, since authorization is case-insensitive.
For those tables that do not comply with the requirements, use the `kudu table rename_table` tool to rename the tables.
3. Create a Ranger client `ranger-kudu-security.xml` configuration file, and note down the directory containing this file.

```
<property>
  <name>ranger.plugin.kudu.policy.cache.dir</name>
  <value>polycache</value>
  <description>Directory where Ranger policies are cached after successful
  retrieval from the Ranger service</description>
</property>
<property>
  <name>ranger.plugin.kudu.service.name</name>
  <value>kudu</value>
  <description>Name of the Ranger service repository storing policies for
  this Kudu cluster</description>
</property>
<property>
  <name>ranger.plugin.kudu.policy.rest.url</name>
```



```

    <value>http://host:port</value>
    <description>Ranger Admin URL</description>
  </property>
  <property>
    <name>ranger.plugin.kudu.policy.source.impl</name>
    <value>org.apache.ranger.admin.client.RangerAdminRESTClient</value>
    <description>Ranger client implementation to retrieve policies from the
    Ranger service</description>
  </property>
  <property>
    <name>ranger.plugin.kudu.policy.rest.ssl.config.file</name>
    <value>ranger-kudu-policymgr-ssl.xml</value>
    <description>Path to the file containing SSL details to connect Ranger A
    dmin</description>
  </property>
  <property>
    <name>ranger.plugin.kudu.policy.pollIntervalMs</name>
    <value>30000</value>
    <description>Ranger client policy polling interval</description>
  </property>

```

4. When Secure Socket Layer (SSL) is enabled for Ranger Admin, add the ranger-kudu-policymgr-ssl.xml file to the Ranger client configuration directory with the following configurations:

```

  <property>
    <name>xasecure.policymgr.clientssl.keystore</name>
    <value>[/path/to/keystore].jks</value>
    <description>Java keystore files</description>
  </property>
  <property>
    <name>xasecure.policymgr.clientssl.keystore.credential.file</name>
    <value>jceks:///file/[path/to/credentials].jceks</value>
    <description>Java keystore credential file</description>
  </property>
  <property>
    <name>xasecure.policymgr.clientssl.truststore</name>
    <value>[/path/to/truststore].jks</value>
    <description>Java truststore file</description>
  </property>
  <property>
    <name>xasecure.policymgr.clientssl.truststore.credential.file</name>
    <value>jceks:///file/[path/to/credentials].jceks</value>
    <description>Java truststore credential file</description>
  </property>

```

5. Set the following configurations on the Kudu master:

```

# The path to directory containing Ranger client configuration. This exa
mple
# assumes the path is '/kudu/ranger-config'.
--ranger_config_path=/kudu/ranger-config

# The path where the Java binary was installed. This example assumes
# '$JAVA_HOME=/usr/local'
--ranger_java_path=/usr/local/bin/java

# The path to the JAR file containing the Ranger subprocess. This example
# assumes '$KUDU_HOME=/kudu'
--ranger_jar_path=/kudu/build/release/bin/kudu-subprocess.jar
# This example ACL setup allows the 'impala' user to access all data st
ored in
# Kudu, assuming Impala will authorize requests on its own. The 'kudu' u
ser is

```

```
# also granted access to all Kudu data, which may facilitate testing and
# debugging (such as running the 'kudu cluster ksck' tool).
--trusted_user_acl=impala,kudu
```

6. Set the following configurations on the tablet servers:

```
--tserver_enforce_access_control=true
```

7. Add a Kudu service repository with the following configurations via the Ranger Admin web UI:

```
# This example setup configures the Kudu service user as a privileged user
# to be
# able to retrieve authorization policies stored in Ranger.

<property>
  <name>policy.download.auth.users</name>
  <value>kudu</value>
</property>
```

Related Information

[Enable Ranger authorization](#)

Ranger client caching

Ranger provides client side cache that use privileges and can periodically poll the privilege store for any changes. When a change is detected, the cache is automatically updated.

Update the `ranger.plugin.kudu.policy.pollIntervalMs` property specified in `ranger-kudu-security.xml` to set how often the Ranger client cache refreshes the privileges from the Ranger service.

Policy for Kudu masters

The following authorization policy is enforced by Kudu masters:

Table 1: Authorization Policy for Masters

Operation	Required Privilege
CreateTable	CREATE ON DATABASE
CreateTable with a different owner specified than the requesting user	ALL ON DATABASE with the Sentry GRANT OPTION.
DeleteTable	DROP ON TABLE
AlterTable (with no rename)	ALTER ON TABLE
AlterTable (with rename)	ALL ON TABLE <old-table> and CREATE ON DATABASE <new-database>
IsCreateTableDone	METADATA ON TABLE
IsAlterTableDone	METADATA ON TABLE
ListTables	METADATA ON TABLE
GetTableLocations	METADATA ON TABLE
GetTableSchema	METADATA ON TABLE
GetTableLocations	METADATA ON TABLE

Policy for Kudu tablet servers

The following authorization policy is enforced by Kudu tablet servers:

Table 2: Authorization Policy for Tablet Servers

Operation	Required Privilege
Scan	SELECT ON TABLE, or METADATA ON TABLE and SELECT ON COLUMN for each projected column and each predicate column
Scan (no projected columns, equivalent to COUNT(*))	SELECT ON TABLE, or SELECT ON COLUMN for each column in the table
Scan (with virtual columns)	SELECT ON TABLE, or SELECT ON COLUMN for each column in the table
Scan (in ORDERED mode)	<privileges required for a Scan> and SELECT ON COLUMN for each primary key column
Insert	INSERT ON TABLE
Update	UPDATE ON TABLE
Upsert	INSERT ON TABLE and UPDATE ON TABLE
Delete	DELETE ON TABLE
SplitKeyRange	SELECT ON COLUMN for each primary key column and SELECT ON COLUMN for each projected column
Checksum	User must be configured in --superuser_acl
ListTablets	User must be configured in --superuser_acl



Note: Unlike Impala, Kudu only supports all-or-nothing access to a table's schema, rather than showing only authorized columns.

Encryption

Kudu allows you to use TLS to encrypt all communications among servers, and between clients and servers.

Configure TLS encryption on Kudu servers using the `--rpc_encryption` flag, which can be set to one of the following options:

- required - Kudu will reject unencrypted connections.
- optional - Kudu will attempt to use encryption, but will allow unencrypted connections.
- disabled - Kudu will not use encryption.

By default, the flag is set to optional. To secure your cluster, set `--rpc_encryption` to required.



Note: Kudu will automatically turn off encryption on local loopback connections, since traffic from these connections is never exposed externally. This allows locality-aware compute frameworks, such as Spark and Impala, to avoid encryption overhead, while still ensuring data confidentiality.

Web UI encryption

The Kudu web UI can be configured to use secure HTTPS encryption by providing each server with TLS certificates. Use the `--webserver_certificate_file` and `--webserver_private_key_file` properties to specify the certificate and private key to be used for communication.

Alternatively, you can choose to completely disable the web UI by setting `--webserver_enabled` flag to false on the Kudu servers.

Redaction

By default all row data is redacted to prevent sensitive data from being included in Kudu server logs or in the web UI. You can turn disable this feature using the `--redact` or the `--webserver_enabled` flag.

Log redaction

To prevent sensitive data from being included in Kudu server logs, all row data is redacted. You can turn off log redaction using the `--redact` flag.

Web UI redaction

To prevent sensitive data from being included in the web UI, all row data is redacted. Table metadata, such as table names, column names, and partitioning information is not redacted. Alternatively, you can choose to completely disable the web UI by setting the `--webserver_enabled` flag to false on the Kudu servers.



Note: Disabling the web UI will also disable REST endpoints such as `/metrics`. Monitoring systems rely on these endpoints to gather metrics data.

Configure a secure Kudu cluster using Cloudera Manager

You can configure a secure Kudu cluster using Cloudera Manager. For that you need enabled Kerberos authentication and RPC encryption, configure coarse-grained authorization, and configure HTTPS encryption. Optionally you can configure fina-grained authorization using Ranger.

Enable Kerberos authentication and RPC encryption

You must already have a secure Cloudera Manager cluster with Kerberos authentication enabled.

Procedure

1. In Cloudera Manager, navigate to **Kudu Configuration**.
2. In the Search field, type Kerberos to show the relevant properties.
3. Find and edit the following properties according to your cluster configuration:

Field	Usage Notes
Kerberos Principal	Set to the default principal, kudu. Currently, Kudu does not support configuring a custom service principal for Kudu processes.
Enable Secure Authentication And Encryption	Select this checkbox to enable authentication and RPC encryption between all Kudu clients and servers, as well as between individual servers. Only enable this property after you have configured Kerberos.

4. Click **Save Changes**.
5. You will see an error message that tells you the Kudu keytab is missing. To generate the keytab, go to the top navigation bar and click **Administration Security**.
6. Go to the **Kerberos Credentials** tab. On this page you will see a list of the existing Kerberos principals for services running on the cluster.
7. Click **Generate Missing Credentials**. Once the **Generate Missing Credentials** command has finished running, you will see the Kudu principal added to the list.

Configure coarse-grained authorization with ACLs

The coarse-grained authorization can be configured with the following two ACLs: the Superuser Access Control List and the User Access Control List. The Superuser ACL is the list of all the superusers that can access the cluster. User-level access can be controlled by using the User ACL. By default, all the users can access the clusters. But when you enable authentication using Kerberos, only the users who are able to authenticate successfully can access the cluster.

Procedure

1. Go to the Kudu service.
2. Click the Configuration tab.
3. Select Category Security .
4. In the Search field, type ACL to show the relevant properties.
5. Edit the following properties according to your cluster configuration:

Field	Usage Notes
Superuser Access Control List	Add a comma-separated list of superusers who can access the cluster. By default, this property is left blank. '*' indicates that all authenticated users will be given superuser access.
User Access Control List	Add a comma-separated list of users who can access the cluster. By default, this property is set to '*'. The default value of '*' allows all users access to the cluster. However, if authentication is enabled, this will restrict access to only those users who are able to successfully authenticate using Kerberos. Unauthenticated users on the same network as the Kudu servers will be unable to access the cluster. Add the impala user to this list to allow Impala to query data in Kudu. You might choose to add any other relevant usernames if you want to give access to Spark Streaming jobs.

6. Click Save Changes.

Enable Ranger authorization

You can configure fine-grained authorization using Apache Ranger. This topic provides the steps to enable Kudu's integration with Ranger from Cloudera Manager.

Procedure

1. From Cloudera Manager, go to Clusters Kudu Configurations .
2. Select the Ranger Service with which Kudu should authorize requests.
3. If Ranger high-availability is enabled for the cluster, add a Kudu service repository with the following configurations through the Ranger Admin web UI is required:

```
# This example setup configures the Kudu service user as a privileged user
# to be
# able to retrieve authorization policies stored in Ranger.

<property>
  <name>policy.download.auth.users</name>
  <value>kudu</value>
```

```
</property>
```

The name of the added Kudu service repository needs to match the one specified in `ranger.plugin.kudu.service.name` of the Ranger client `ranger-kudu-security.xml` configuration file.



Note: When a Kudu client opens a table, the Kudu Master will authorize all possible actions the user may want to perform on the given table (ALL, and if it's not allowed, then INSERT, SELECT, UPDATE, DELETE). This results in auditing these requests when a client opens a table, even if they'll never do any of these operations.

Related Information

[Configure Kudu's integration with Apache Ranger](#)

Configure HTTPS encryption

Lastly, you enable TLS/SSL encryption (over HTTPS) for browser-based connections to both the Kudu master and tablet server web UIs.

Procedure

1. Go to the Kudu service.
2. Click the Configuration tab.
3. Select Category Security .
4. In the Search field, type TLS/SSL to show the relevant properties.
5. Edit the following properties according to your cluster configuration:

Field	Usage Notes
Master TLS/SSL Server Private Key File (PEM Format)	Set to the path containing the Kudu master host's private key (PEM-format). This is used to enable TLS/SSL encryption (over HTTPS) for browser-based connections to the Kudu master web UI.
Tablet Server TLS/SSL Server Private Key File (PEM Format)	Set to the path containing the Kudu tablet server host's private key (PEM-format). This is used to enable TLS/SSL encryption (over HTTPS) for browser-based connections to Kudu tablet server web UIs.
Master TLS/SSL Server Certificate File (PEM Format)	Set to the path containing the signed certificate (PEM-format) for the Kudu master host's private key (set in Master TLS/SSL Server Private Key File). The certificate file can be created by concatenating all the appropriate root and intermediate certificates required to verify trust.
Tablet Server TLS/SSL Server Certificate File (PEM Format)	Set to the path containing the signed certificate (PEM-format) for the Kudu tablet server host's private key (set in Tablet Server TLS/SSL Server Private Key File). The certificate file can be created by concatenating all the appropriate root and intermediate certificates required to verify trust.
Enable TLS/SSL for Master Server	Enables HTTPS encryption on the Kudu master web UI.
Enable TLS/SSL for Tablet Server	Enables HTTPS encryption on the Kudu tablet server Web UIs.

6. Click Save Changes.

Configure a secure Kudu cluster using flag safety valves

You should set the configuration parameters on all the servers (master and tablet servers) to ensure that a Kudu cluster is secure.



Important: Follow these command-line instructions on systems that do not use Cloudera Manager. If you are using Cloudera Manager, see *Configure a secure Kudu cluster using Cloudera Manager*.

```
# Connection Security
#-----
--rpc_authentication=required
--rpc_encryption=required
--keytab_file=<path-to-kerberos-keytab>
# Web UI Security
#-----
--webserver_certificate_file=<path-to-cert-pem>
--webserver_private_key_file=<path-to-key-pem>
# optional
--webserver_private_key_password_cmd=<password-cmd>

# If you prefer to disable the web UI entirely:
--webserver_enabled=false

# Coarse-grained authorization
#-----
# This example ACL setup allows the 'impala' user as well as the
# 'etl_service_account' principal access to all data in the
# Kudu cluster. The 'hadoopadmin' user is allowed to use administrative
# tooling. Note that by granting access to 'impala', other users
# may access data in Kudu via the Impala service subject to its own
# authorization rules.
--user_acl=impala,etl_service_account
--admin_acl=hadoopadmin
```

More information about these flags can be found in the configuration reference documentation.

Related Information

[Apache Kudu Configuration Reference](#)

[Configure a secure Kudu cluster using Cloudera Manager](#)