Managing Apache Kudu Security

Date published:
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Kudu security considerations

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

Kudu uses strong authentication with Kerberos, and authorization with Ranger. Communication between Kudu clients and servers, and between servers to other servers, can be encrypted with TLS. Kudu also supports HTTPS on 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.

With support of JWT for authenticating Kudu clients and RPC forwarding through a TCP proxy, more options are available for deploying secure Kudu clusters. For example, you can deploy a secure (for example, Kerberised) Kudu cluster in an internal or firewalled network and allow Kudu clients running outside of the network to connect to the cluster by authenticating themselves through JWT to Kudu servers, where all the RPC traffic is forwarded through a TCP proxy. In such a deployment, all the communications between Kudu clients running outside of the internal network and the Kudu cluster running in the internal network are protected by TLS, where the clients are provided with the cluster's IPKI CA certificate to authenticate the server side they are communicating with.

Proxied RPCs in Kudu

You can configure RPC endpoints for Kudu servers to separate internal and external traffic in a Kudu cluster to enable inter-cluster communications.

You can separate the internal and the external traffic in a Kudu cluster while providing the connectivity for Kudu clients running in external networks where the internal traffic is never routed through a proxy's or a loadbalancer's endpoint. Essentially, it allows for the internal traffic (for example, the traffic between tablet servers and masters) to bypass advertised RPC addresses, using alternative addresses for inter-cluster communications.

You can configure additional RPC endpoints for Kudu servers to handle traffic proxied from external networks. So, when a Kudu server receives an RPC request, it possesses enough information to decide whether to handle the request as arriving from an internal network or an external network. All the communications of Kudu components within the cluster's internal network are routed through the standard RPC endpoints. But the requests proxied from external networks are routed through those dedicated RPC endpoints.When a Kudu server receives an RPC request through such an endpoint, it sends a response where the internal RPC addresses of Kudu servers are substituted with corresponding RPC addresses reachable to Kudu clients outside the cluster through a TCP proxy.

To support proxied RPCs, the following flags are introduced; both flags accepting comma-separated list of strings in the <hostname>:<port> form:

- --rpc_proxy_advertised_addresses

  Sets the server's RPC endpoints exposed to the external network through a TCP proxy.
- --rpc_proxied_addresses

  Defines RPC endpoints in the internal network to handle RPC requests forwarded or proxied from external networks. It is possible to use a wild card for IP address (for example, 0.0.0.0) and the port number (for example, 0) for the elements of this address list.

The --rpc_proxy_advertised_addresses flag is orthogonal to the --rpc_advertised_addresses flag, so it is possible to use both simultaneously. For example, it might be useful if the network environment for docker containers in the private internal network is configured in some special way that mandates using the RPC-advertised addresses instead of the real RPC addresses for Kudu servers.

Kudu security limitations

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

- Kudu uses an internal PKI system to issue X.509 certificates to servers in the cluster. As a result, you cannot run Kudu with public IPs.
- Server certificates generated by Kudu IPKI are incompatible with bouncycastle version 1.52 and earlier.
- 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 fails. But that does not mean that the service is not working.

Kudu 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.

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. 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.

Kudu authentication with Kerberos

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

The --rpc_authentication flag is used to configure authentication on Kudu servers. In a Cloudera Manager managed cluster that flag is configured using the Enable Secure Authentication And Encryption property. This property can have the following values:

- Optional: Kudu will attempt to use strong authentication, but will allow unauthenticated connections. The connection between servers will be encrypted. This is the default value which is indicated by a clear checkbox.
- Required: Kudu will reject connections from clients and servers who lack authentication credentials. If you want to secure your cluster you have to set the --rpc_authentication flag to this value, meaning that you have to select the Enable Secure Authentication And Encryption property.

Kudu 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.COM

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

JWT authentication for Kudu

JSON Web Token (JWT) authentication is a standard way of securely transmitting signed information between two parties. Learn how Kudu clients can use JWT to authenticate to a Kudu cluster.

JWT is a compact, URL-safe mechanism of representing claims to be transferred between two parties (for example, Kudu server and Kudu client). The claims in a JWT token are encoded either as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext format of a JSON Web Encryption (JWE) structure. This encoding enables the claims to be digitally signed, integrity protected with a Message Authentication Code (MAC), and/or encrypted. The structure of JWT enables you to verify whether the content is tampered.

Kudu servers accept JWTs for authentication when run with the --enable_jwt_token_auth flag (by default, it is set to false). Correspondingly, all servers in a Kudu cluster should be able to access the JWT issuer's JSON Web Key Set (JWKS) to verify the JWTs presented to them by connecting clients. Kudu servers fetch JWKS from the location specified by the --jwks_url flag.

When to use JWT authentication

JWT authentication is an alternative to Kerberos authentication, and you can use it in situations where Kerberos authentication is not a viable option but authentication is required, as in the following examples,

- A Kudu client runs on a node outside the Kerberos realm used in the cluster's internal network, and cross-realm authentication cannot be configured.
- A Kudu client cannot access Kerberos servers in the Kudu cluster's internal network, but it can talk to those Kudu servers through a TCP proxy.
- A Kudu client runs on a node without any Kerberos environment configured.

How JWT authentication works

In Kudu, a client sends a signed and non-encrypted JWT to the Kudu server when negotiating a new RPC connection between them. The JWTs are sent over communication channels protected by TLS. The Kudu servers check the

validity of the JWTs by verifying their signatures. In its turn, a Kudu client confirms the authenticity of a Kudu server by verifying the signature of the server's TLS certificate. So, for JWT-based authentication scenarios in Kudu, TLS certificates used by servers should be trusted by clients. If Kudu is deployed with its own certificate authority (that is a part of the Kudu's Internal Private Key Infrastructure, or IPKI), the Kudu's IPKI CA certificate should be added as trusted by using certain Kudu client API calls. For example, use KuduClientBuilder::trusted_certificate(const std::st ring&) in C++ Kudu applications to add a certificate in PEM format in the client's chain of trusted certificates. The Kudu's IPKI CA certificate can be retrieved from the /ipki-ca-cert HTTP/HTTPS endpoint of the embedded master's web server.

Configuring server side JWT authentication for Kudu

Learn how to configure server-side changes for JSON Web Token (JWT) authentication for Kudu.

Before you begin

JWT authentication is an experimental feature. To change the flags mentioned in the following steps, you need to add --unlock_experimental_flags to the Kudu safety valve configuration.

Procedure

1. Log in to Cloudera Manager.
2. Select the Kudu service.
3. Click Configurations.
4. Search for Kudu Service Advanced Configuration Snippet (Safety Valve) for gflagfile, and configure the required configuration parameters for JWT authentication:

   ```
   --unlock_experimental_flags
   --enable_jwt_token_auth=true
   --jwks_url=...
   ```

   The following list describes the configuration flags available:

   enable_jwt_token_auth

   Valid option: Boolean

   Default value: false

   Description: This enables JWT authentication. The server expects a valid JWT to be sent by the client which is verified when the connection is being established. When true, reads the JWT token out of the RPC and extracts user name from the token payload.

   Either --jwks_file_path or --jwks_url (but not both) must be set when --enable_jwt_token_auth is set to true.

   jwks_file_path

   Valid option: String

   Description: File path of the pre-installed JSON Web Key Set (JWKS) for JWT verification.

   jwks_url

   Valid option: String

   Description: URL of the JWKS for JWT verification.
5. Add --trusted_certificate_file=... flag as well, if the JWKS server you are using does not have an SSL certificate signed by a CA that is trusted on the system level.

Configuring client side JWT authentication for Kudu

Learn how to configure client-side changes for JSON Web Token (JWT) authentication for Kudu.

Procedure

Set JWT and trusted certificate:

For C++ Client

```
 return KuduClientBuilder()
      .master_server_addrs(master_addrs)
      .default_admin_operation_timeout(MonoDelta::FromSeconds(20))
      .jwt(cJwt)
      .trusted_certificate(cKuduIpkiCaCert)
      .Build(client);
}
```

For Python Client

```
client = kudu.connect(self.master_hosts,
                              self.master_ports,
                              require_authentication=True,
                              jwt=jwtToken,
                              trusted_certificates=certs)
```

For Java Client

```
KuduClient client = new KuduClient.KuduClientBuilder(KUDU_MASTERS).build
();
client.jwt(jwtToken);
client.trustedCertificates(Arrays.asList(ByteString.copyFrom(ipkiCaCertDer
Bytes)));
```

Note: The Java client expects the trusted certificate to be in DER format, while the C++ client expects it to be in PEM format. The master server's Internal Private Key Infrastructure (IPKI) certificates are accessible on its webserver's /ipki-ca-cert-der and /ipki-ca-cert endpoints respectively.

Kudu 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.

Kudu fine-grained authorization

Kudu can be configured to enforce fine-grained authorization across servers.

Fine-grained authorization ensures that users can see only the data they are explicitly authorized to see. Kudu supports this by leveraging policies defined in Apache Ranger.

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

Kudu and Apache Ranger integration

Learn about how Apache Ranger is integrated with Kudu in order to provide fine-grained authorization across servers.

The 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.

Table creation requires CREATE ON DATABASE privilege. If the user creates a table with a different owner, ALL and delegate admin are required.

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

Kudu authorization tokens

Learn about authorization tokens which are used by Ranger to propagate and check privileges.

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 limit 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).

Specifying trusted users

You can specify which users can view and modify data stored in Kudu. Additionally, some services that interact with Kudu may authorize requests on behalf of their end users.

About this task

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.

Furthermore, some services that interact with Kudu can 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.

Procedure

1. In Cloudera Manager, navigate to Kudu Configuration .
2. Find the Master Advanced Configuration Snippet (Safety Valve) for gflagfile propertz.
3. Click View as XML and add the following configuration:

```
--trusted_user_acl=impala,hive,kudu,rangeradmin,[***TRUSTED USER***]
```

4. Click Save Changes.
5. Restart the Kudu service.

Kudu authorization policies

Review the authorization policies that are enforced by Kudu masters and Kudu tablet servers.

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
GetTabletLocations	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 COLUMNfor 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.

Ranger policies for Kudu

There are two Kudu related Ranger policies which are applied based on how you are accessing Kudu.

There are two resource-based services in Ranger that are used in relation to Kudu: cm_kudu and Hadoop SQL.

The Kudu service and its connected clients, such as Spark, native C++, and Java clients, use the cm_kudu resource-based service.

The Hadoop SQL resource-based service is used by Hive and Impala when Kudu is accessed through them.

When Kudu is accessed by Impala, the Impala service performs actions as the impala user in Kudu. The impala user is set as a trusted user in Kudu, meaning that privilege checks are completely bypassed and the impala user is granted full access. As a result, the cm_kudu resource-based service is not applied, only the Hadoop SQL resource-based service is used to check for permission and privileges.

As a result, when you are accessing Kudu through Hive or Impala, you must ensure that all applicable permission and privileges are configured in the Hadoop SQL resource-based service.

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

Disabling redaction

Data redaction is used to prevent sensitive data from being included in Kudu server logs or in the web UI. Data redaction is enabled by default but you can disable it 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.

Procedure

1. In Cloudera Manager, navigate to Kudu Configuration .
2. Find the Kudu Service Advanced Configuration Snippet (Safety Valve) for gflagfile property.
3. If you want to disable log redaction, add the following configuration:

```
--redact=false
```

4. If you want to Web UI disable redaction, add the following configuration:

```
--webserver_enabled=false
```

5. Click Save Changes.
6. Restart the Kudu service.

Configuring 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 custom Kerberos principal, TLS/SSL encryption or fine-grained authorization using Ranger.

Enabling 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. If this is not selected, security is not enforced but secured clients are not rejected either and the connection will be encrypted if both sides support it.

4. Click Save Changes.
   An error message displayed that tells you the Kudu keytab is missing.
5. Navigate to Administration Security .
6. Select 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.

Configuring custom Kerberos principal for Kudu

You can configure a custom Kerberos principal for Kudu using Cloudera Manager.

Procedure

1. In Cloudera Manager, navigate to Kudu Configuration .
2. Search for kerberos.
3. Find the Kerberos principal property and set it to the required name.
4. Click Save changes.
   Custom Kerberos principal for Kudu is applied for not just the Kudu server but also for the Hive and Impala clients.
5. For components that are not controlled by Cloudera Manager, configure the SASL protocol name.

   You must add the SASL protocol name configuration wherever the Kudu client properties, such as master addresses, are configured.

   For example, if the principal name is kudu/_HOST set the protocol to kudu.

Configuring 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.

Field	Usage Notes
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.

Configuring TLS/SSL encryption for Kudu using Cloudera Manager

TLS/SSL encryption is enabled between Kudu servers and clients by default. You can enable TLS/SSL encryption for Kudu web UIs or configure the encryption using Cloudera Manager.

Procedure

1. In Cloudera Manager, navigate to Kudu Configuration .
2. In the Search field, type TLS/SSL to show the relevant properties.
3. Edit the following properties according to your cluster configuration:

Table 3: TLS/SSL Kudu properties

Property	Description
Master TLS/SSL Server Private Key File (PEM Format)	Set to the path containing the Kudu master host's private key (Privacy Enhanced Mail (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.
Enable Secure Authentication, Encryption, and Web UI	Enables Kerberos for Kudu servers, clients and web servers, and enables encryption for Kudu servers and clients.

4. Click Save Changes.

Enabling 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, UDPATE, DELETE). This results in auditing these requests when a client opens a table, even if they'll never do any of these operations.

Configuring 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.

Configuring data at rest encryption

You can enable data at rest encryption using Cloudera Manager. However, you can enable it only for a fresh installation and once Kudu directories exist on the cluster you cannot disable the encryption.

About this task

Kudu supports data at rest encryption, specifically AES-128, AES-192 and AES-256 ciphers in CTR mode are supported to encrypt data.

When data is encrypted at rest, the following keys are used for encryption:

- File Key: The unique key used to encrypt a physical file
- Server Key: A server's own key used to encrypt the File Key
- Cluster Key: A key that is used to encrypt the Server Key and stored in a third-party Key Management Service (KMS). Kudu supports Apache Ranger KMS and Apache Hadoop KMS. Both are API-compatible

Before you begin

If Kudu was started before encryption was enabled, all Kudu data directories, WAL directories, and metadata directories (usually the same as WAL directories) must be deleted. This will result in a completely clean slate, so if there is any data in Kudu, it must be backed up using the backup tool before doing so. For more information, see *Kudu backup*.

Procedure

1. In Cloudera Manager navigate to Kudu Configuration .
2. Search for data at rest.
3. Find the Encrypt data at rest property and select it to enabe data at rest encryption.
4. Find the Data at rest encryption cipher property and select the cipher for the encryption.

   Its defaault value is AES-128-CTR.
5. Find the Data at rest encryption key name property and set its value to the name of the data at rest encryption key.

   Its default value is kudu_encryption_key.
6. Create a key in Ranger KMS with the same name as the value of the Data at rest encryption key name property.

   For more information, see *List and create keys*.
7. Ensure that the TLS certificate used by Ranger KMS is trusted on the OS-level by all Kudu hosts. This is required even if you use AutoTLS.

   For more information, consult your OS documentation.
8. Start the Kudu cluster.

What to do next

If you enabled data at encryption for a pre-existing cluster, restore the data you backed up before enabling encryption. For more information, see *Kudu recovery*.

Related Information

Kudu backup

List and create keys

Kudu recovery