

Trino federation and connectors

Date published:

Date modified:

The Cloudera logo, featuring the word "CLOUDERA" in a bold, orange, sans-serif font. The letter "E" is stylized with a horizontal bar through its center.

Legal Notice

© Cloudera Inc. 2026. 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

- Trino connectors.....4**
 - Apache Kudu connector support for Trino.....4
 - Configuring the Apache Kudu connector.....4
 - Connector configuration.....4
 - Querying data using Trino.....5
 - Datatype mappings.....6
 - SQL support.....7
 - Kudu Limitations and roadmap.....8
- Support for Teradata connector.....9**
- Configuring connection pooling for JDBC-based connectors..... 10**

Trino connectors

Trino is a distributed SQL query engine that provides connectors to connect to and access data from various remote data sources. Cloudera supports a range of certified Trino connectors for accessing data across your infrastructure.

For detailed information about available Trino connectors and configuration options, see the child topics.

Apache Kudu connector support for Trino

The Apache Kudu connector enables querying, inserting, and deleting data in Apache Kudu using Trino.

Apache Kudu connector enables querying, inserting, and deleting data in Apache Kudu using Trino clusters.

Apache Kudu connector requires:

- Apache Kudu version 1.13.0 or higher
- Trino version 2026.0.24.0-30 or higher.
- Network access from the Trino coordinator and worker nodes to the Apache Kudu cluster on default port 7051.

For more information, see the *Apache Kudu* documentation.

Related Information

[Apache Kudu](#)

Configuring the Apache Kudu connector

Configure the Apache Kudu connector to stream data between Trino nodes and your Apache Kudu cluster.

About this task

You can source data directly from Apache Kudu into your target environment by using the Apache Kudu connector. The connector retrieves data directly from the storage engine while handling network access implicitly.

Before you begin

- Verify network access exists between the Trino nodes and the Apache Kudu cluster on port 7051.

Procedure

1. Create a catalog properties file named `kudu.properties` in the `etc/catalog/` directory.
2. Add the required configuration properties to the file.

Properties

`connector.name=kudu`

`kudu.client.master-addresses=kudu.example.com:7051`

3. Save the file and restart the Trino service.

Connector configuration

Apache Kudu connector configuration reference provides the properties and default values to customize the connector settings.

Kudu connector configuration

The following table shows the configuration properties for the Apache Kudu connector:

Property name	Description	Default
kudu.authentication.type	Specifies the authentication type. Valid values are NONE or KERBEROS.	NONE
kudu.client.master-addresses	Specifies a comma-separated list of Kudu master addresses. Supported formats: example.com, example.com:7051, 192.0.2.1, 192.0.2.1:7051,[2001:db8::1], [2001:db8::1]:7051, 2001:db8::1	
kudu.schema-emulation.enabled	Enables schema emulation to map Kudu table prefixes to Trino schemas.	false
kudu.schema-emulation.prefix	Specifies the prefix to use for schema emulation.	presto::
kudu.client.default-admin-operation-timeout	Specifies the default timeout duration for administrative operations.	30s
kudu.client.default-operation-timeout	Specifies the default timeout duration for user operations.	30s
kudu.client.disable-statistics	Disables statistics collection by the Kudu client.	false
kudu.allow-local-scheduling	Assigns Kudu splits to a replica host if the worker and Kudu share the host.	false

Kerberos authentication properties

The following table shows the Kerberos authentication properties when kudu.authentication.type is set to KERBEROS:

Property name	Description	Default
kudu.authentication.client.principal	Specifies the Kerberos client principal name.	
kudu.authentication.client.keytab	Specifies the path to the Kerberos keytab file.	
kudu.authentication.config	Specifies the path to the krb5.conf file.	
kudu.authentication.server.principal.primary	Specifies the custom SPN if Kudu runs with one.	kudu

Querying data using Trino

Learn how to query data in Apache Kudu using Trino with or without schema emulation.

Apache Kudu does not support namespaces or schemas for tables. The Cloudera Trino Kudu connector manages schemas in Apache Kudu clusters by providing optional schema emulation through table naming conventions.

Default behavior without schema emulation

Schema emulation is disabled by default. In this mode, all Apache Kudu tables reside in the default schema:

- Query an Apache Kudu table named orders in Trino by running `SELECT * FROM example.default.orders` or `SELECT * FROM orders` when catalog and schema are set to kudu and default.
- Enclose table names in double quotes when they contain special characters, for example, `SELECT * FROM example.default."special.table!"`.

The following example shows how to create a table, inspect the table structure, insert data, and query the results in the default schema:

Create the users table:

```
CREATE TABLE example.default.users (
  user_id int WITH (primary_key = true),
  first_name VARCHAR,
  last_name VARCHAR
```

```
) WITH (
  partition_by_hash_columns = ARRAY['user_id'],
  partition_by_hash_buckets = 2
);
```

Describe the table structure:

```
DESCRIBE example.default.users;
```

The command returns the following column metadata:

Column	Type	Extra
user_id	integer	primary_key, encoding=auto, compression=default
first_name	varchar	nullable, encoding=auto, compression=default
last_name	varchar	nullable, encoding=auto, compression=default

Insert data into the table:

```
INSERT INTO example.default.users VALUES (1, 'Donald', 'Duck'), (2,
'Mickey', 'Mouse');
```

Query the inserted data:

```
SELECT * FROM example.default.users;
```

Behavior with schema emulation

When schema emulation is enabled in the catalog configuration file (for example, etc/catalog/example.properties), the connector maps Apache Kudu table names to Trino schemas.

When `kudu.schema-emulation.enabled=true` and `kudu.schema-emulation.prefix=`, the connector uses the following mappings and creates a special table named `$schemas` to manage schemas:

Kudu table name	Trino qualified name
orders	kudu.default.orders
part1.part2	kudu.part1.part2
x.y.z	kudu.x."y.z"

When `kudu.schema-emulation.enabled=true` and `kudu.schema-emulation.prefix=presto::`, the connector uses the following mappings and creates a special table named `presto::$schemas` to manage schemas:

Kudu table name	Trino qualified name
orders	kudu.default.orders
part1.part2	kudu.default."part1.part2"
x.y.z	kudu.default."x.y.z"
presto::part1.part2	kudu.part1.part2
presto:x.y.z	kudu.x."y.z"

Datatype mappings

Apache Kudu connector data type mappings reference provides the bidirectional data type conversions between Apache Kudu and Trino.

Data type mappings between Apache Kudu and Trino

The following table shows the data type mappings between Apache Kudu and Trino:

Kudu type	Trino type	Notes
BOOL	BOOLEAN	Maps bidirectionally.
INT8	TINYINT	Maps bidirectionally.
INT16	SMALLINT	Maps bidirectionally.
INT32	INTEGER	Maps bidirectionally.
INT64	BIGINT	Maps bidirectionally.
FLOAT	REAL	Maps bidirectionally.
DOUBLE	DOUBLE	Maps bidirectionally.
DECIMAL(p,s)	DECIMAL(p,s)	Supported for Kudu server version 1.7.0 or higher.
STRING	VARCHAR	Optional maximum length is lost when mapping to Kudu.
BINARY	VARBINARY	Maps bidirectionally.
DATE	DATE	Maps bidirectionally.
UNIXTIME_MICROS	TIMESTAMP(3)	Microsecond resolution in Kudu is reduced to millisecond resolution in Trino.

SQL support

Apache Kudu connector SQL support reference provides the supported SQL statements, table properties, partitioning options, column properties, and procedures for querying and managing Kudu data through Trino.

Supported SQL statements

The connector provides read and write access to data and metadata in Apache Kudu. In addition to standard read operations, the connector supports the following statements:

- **INSERT** - Functions as an UPSERT operation for **INSERT INTO ... VALUES** and **INSERT INTO ... SELECT**.
- **DELETE** - Requires a **WHERE** clause with predicates that can be fully pushed down to the data source.
- **MERGE** - Performs conditional insert, update, or delete operations.
- **CREATE TABLE** - Creates a table in Kudu with explicit primary key and partitioning definitions.
- **CREATE TABLE AS** - Creates a table and populates the table with query results.
- **DROP TABLE** - Removes the table from Kudu.
- **ALTER TABLE** - Modifies the table structure. Adding columns supports column properties. Modifying or dropping columns is restricted to non-primary-key columns.
- **CREATE SCHEMA** - Allowed only when schema emulation is enabled.
- **DROP SCHEMA** - Allowed only when schema emulation is enabled.

Creating a table

Creating an Apache Kudu table requires specifying column names, data types, primary keys, and partitioning design. Option settings include column encoding, compression, and replication factor.

```
CREATE TABLE user_events (
  user_id INTEGER WITH (primary_key = true),
  event_name VARCHAR WITH (primary_key = true),
  message VARCHAR,
  details VARCHAR WITH (nullable = true, encoding = 'plain')
)
```

```
WITH (
  partition_by_hash_columns = ARRAY['user_id'],
  partition_by_hash_buckets = 5,
  number_of_replicas = 3
);
```

The table structure adheres to the following rules:

1. Primary key columns must be listed first in the column list.
2. All columns specified in partition definitions must belong to the primary key.
3. The optional `number_of_replicas` table property defines the number of tablet replicas and must be an odd integer. If omitted, the Kudu master default replication factor applies.

Column properties

The following table shows the column properties available when creating or altering tables:

Column property name	Type	Description
<code>primary_key</code>	BOOLEAN	Specifies whether the column belongs to the primary key. Apache Kudu primary keys enforce uniqueness. Inserting duplicate primary keys updates existing rows.
<code>nullable</code>	BOOLEAN	Specifies whether the column accepts null values. Primary key columns cannot be nullable.
<code>encoding</code>	VARCHAR	Specifies the column encoding to optimize storage and performance. Valid values: <code>auto</code> , <code>plain</code> , <code>bitshuffle</code> , <code>runlength</code> , <code>prefix</code> , <code>dictionary</code> , <code>group_varint</code> .
<code>compression</code>	VARCHAR	Specifies the compression algorithm for encoded values. Valid values: <code>default</code> , <code>no</code> , <code>lz4</code> , <code>snappy</code> , <code>zlib</code> .

The following example configures specific encoding and compression properties on columns:

```
CREATE TABLE example_table (
  name VARCHAR WITH (primary_key = true, encoding = 'dictionary', compression = 'snappy'),
  index BIGINT WITH (nullable = true, encoding = 'runlength', compression = 'lz4'),
  comment VARCHAR WITH (nullable = true, encoding = 'plain', compression = 'default'),
  ...
) WITH (...);
```

Modifying tables

Add a column to an existing table by using the `ALTER TABLE ... ADD COLUMN` statement:

```
ALTER TABLE example_table ADD COLUMN extraInfo VARCHAR WITH (nullable = true, encoding = 'plain')
```

Renaming or dropping columns using `ALTER TABLE ... RENAME COLUMN` or `ALTER TABLE ... DROP COLUMN` is allowed only for columns that are not part of the primary key.

Kudu Limitations and roadmap

Overview of supported features, current limitations, and planned future enhancements for Apache Kudu.

Apache Kudu supports lowercase table names, application-level default column values, non-managed statistics, Hive Metastore integration, and custom scanner configurations.

Limitations

The following limitations apply when working with Kudu:

- Only lowercase table and column names in Kudu are supported.
- Adding default column values is not supported.
- Collecting managed statistics by using the ANALYZE command is not supported.

Future tasks

The following features and improvements are planned for future releases:

- Supporting Kudu Hive Metastore (HMS) integration for schema emulation.
- Supporting Kudu scanner-specific configurations.

Support for Teradata connector

Cloudera Data Warehouse offers support for a read-only Trino-Teradata connector designed to facilitate SELECT operations on Teradata sources. It operates strictly in ANSI Mode, optimizing performance by pushing down filters and aggregates directly to the source database.



Note: This feature is in technical preview and not recommended for use in production deployments. Cloudera recommends that you try this feature in test and development environments.

Key features and capabilities

Supported data type

The connector includes comprehensive type mappings for:

- Numeric: TINYINT, SMALLINT, INTEGER, BIGINT, REAL, DOUBLE, DECIMAL(p, s)
- String: CHAR, VARCHAR (bounded and unbounded)
- Date/Time: DATE, TIMESTAMP (with or without Time Zone, precision up to 6)

Constraints related to TIMESTAMP data type

The core problem is the inability of the traditional TIMESTAMP data type to store time zone information, a limitation found in both databases and the legacy `java.sql.Timestamp` class. While modern Java introduces classes like `OffsetDateTime` and `ZonedDateTime` to handle time zones, the official JDBC 4.2 specification still relies on the older `java.sql.Timestamp`. Teradata attempts to support the modern `OffsetDateTime` through its driver's `setObject` method, but this implementation has issues, as it internally reverts to legacy `java.sql.Timestamp` and Calendar conversions, reintroducing the original time zone issues.

Workaround: The newer SQL standard introducing the `TIMESTAMP WITH TIME ZONE` data type is supported by many vendors, including Trino and Teradata. The primary goal of this type is to ensure that constant literals and prepared statements retain their associated time zone upon insertion, making the database immune to changes in system or JVM default time zone settings. This functionality is essential for cross-region deployments, guaranteeing that data can be accurately queried and client applications can perform reliable time zone translations based on the captured time zone.

Query optimization

To balance performance with Teradata's architectural constraints, pushdown logic is applied selectively:

- Fully Pushed Down:
 - Numeric, Date, and Time predicates.

- Basic aggregates: count(), min(), max(), sum(), and avg().
- Not Pushed Down:
 - Character-based inequalities.

System features

- Connection Management: Features integrated JDBC connection pooling.
- Metadata Caching: Configurable caching for schemas and tables to reduce overhead.
- Naming Conventions: Enforces case-insensitive name matching for schemas and tables, complying with Teradata's native uniqueness constraints.

Limitations and unsupported operations

This connector is strictly read-only. Attempting any of the following will trigger a NOT_SUPPORTED exception:

- DML Operations: INSERT, UPDATE, DELETE
- DDL Operations: DROP, TRUNCATE, RENAME

Configuring connection pooling for JDBC-based connectors

Learn how to configure connection pooling capabilities for JDBC-based Trino connectors. Connection pooling helps in better performance, resource utilization, and stability while querying different data sources using Trino.

About this task

Currently, a new database connection is created for every query or sub-query, which causes several inefficiencies:

- Performance Overhead: Significant latency from repeatedly setting up and tearing down connections.
- Resource Exhaustion: Rapidly opening and closing connections can deplete the database server resources, leading to failures.
- Scalability Challenges: The overhead of managing individual connections becomes a bottleneck as the number of users and queries grows.

Connection pooling is currently supported for the following JDBC-based connectors:

- Hive-jdbc
- Impala
- MySQL
- PostgreSQL
- MariaDB
- Teradata
- Oracle



Note: To configure connection pooling for the Oracle connector, please refer to the configuration parameters provided in upstream Trino documentation.

Procedure

1. Log in to the Cloudera Data Warehouse service and click Federation Connectors.
The **Federation Connectors** page is displayed that lists all the currently configured connectors.
2. For the JDBC-based connector for which you want to enable connection pooling, click  Edit Configuration .
3. In the **General Details** tab, add the following connection pooling configurations as key-value pairs in the **Configuration** section:

Key	Description	Default Value
connection-pool.enabled	Enables or disables connection pooling for the connector.	false
connection-pool.max-size	Specifies the maximum number of connections in the pool.	30
connection-pool.max-connection-lifetime	Specifies the maximum lifetime (in minutes) of a connection in the pool.	30m
connection-pool.pool-cache-max-size	Specifies the maximum number of data sources cached in memory.	1000
connection-pool.pool-cache-ttl	Specifies the duration (in minutes) after which an inactive data source is dropped.	30m



Note: The connection pool is configured at the node level. This means the total number of connections to the data source is calculated as `connection-pool.max-size` X number of nodes in the cluster. You must consider this total when setting the value to ensure that the underlying data source can support it. For example, 30 max-size on 10 nodes allows up to 300 total connections.

4. Click Apply Changes.