Data Lake Compute

Client Access

Copyright Notice

Trademark Notice

Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Client Access
Accessing JDBC
DLC JDBC Access

Last updated: 2026-06-03 11:26:22

Supported Engine Types

- SuperSQL Engine
- Standard Spark Engine
- Standard Presto Engine

> **Note:**
> The Presto engine is deprecated and only available for existing users.

Preparing the Environment

- Dependency: JDK 1.8
- JDBC Download: Click to download the JDBC driver.

Connecting to DLC

1. Load the DLC JDBC driver.

```
Class.forName("com.tencent.cloud.dlc.jdbc.DlcDriver");
```

2. Create a Connection through DriverManager.

```
Connection cnct = DriverManager.getConnection(url, secretId,
secretKey);
```

URL Format

```
jdbc:dlc:dlc.tencentcloudapi.com?
task_type=SQLTask&database_name=abc&datasource_connection_name=DataLakeC
atalog&region=ap-nanjing&data_engine_name=spark-
cu&result_type=COS&read_type=Stream
```

JDBC Connection String Parameter Description:

Parameter	Required	Description
dlc_endpoint	Yes	The Endpoint for the DLC service, which is fixed as dlc.tencentcloudapi.com.
datasource_connection_name	Yes	The name of the data source connection, which corresponds to the DLC data catalog.
task_type	Yes	Task type. • Presto engine: Enter SQLTask. • For the SparkSQL engine or the standard engine SQL resource group, enter: SparkSQLTask. • For the Spark job engine or the standard engine batch job, enter: BatchSQLTask.
database_name	No	Database name.
region	Yes	Region. Currently, the DLC service supports ap-nanjing, ap-beijing, ap-guangzhou, ap-shanghai, ap-chengdu, ap-chongqing, na-siliconvalley, ap-singapore, and ap-hongkong.
data_engine_name	Yes	Data Engine Name
secretId	Yes	The SecretId in TencentCloud API Key Management
secretKey	Yes	The SecretKey in TencentCloud API Key Management
result_type	No	The default value is Service. If you require faster result retrieval, you can set it to COS. • Service: Obtain results through the DLC API. • COS: Obtain results through the COS client (not supported by the standard engine Presto).
read_type	No	• Stream: Obtain results from COS in a streaming manner. • DownloadSingle: Download a single file to your local machine to obtain the result.

		The default value is Stream. This value is meaningful only when result_type is COS. The download location is the /tmp temporary directory. Ensure that the directory has read and write permissions. The data will be automatically deleted after it is read.
resource_group_name	No	Standard Spark Engine Resource Group Name

Execute the query

```
Statement stmt = cnct.createStatement();ResultSet rset =
stmt.executeQuery("SELECT * FROM dlc");
while (rset.next())
 {// process the results
}
rset.close();
stmt.close();
conn.close();
}
rset.close();
stmt.close();
conn.close();
```

Syntax Support

The syntax available for JDBC is consistent with the DLC standard syntax.

Sample Code

Database and Table Operations

```
import java.sql.*;
public class MetaTest {
        public static void main(String[] args) throws SQLException {
            try {
                Class.forName("com.tencent.cloud.dlc.jdbc.DlcDriver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            return;
    }
```

```
    Connection connection = DriverManager.getConnection(
                    "jdbc:dlc:<dlc_endpoint>?task_type=
<task_type>&database_name=
<database_name>&datasource_connection_name=DataLakeCatalog&region=
<region>&data_engine_name=<data_engine_name>&result_type=<result_type>",
                    "<secret_id>",
                    "secret_key");
    Statement statement = connection.createStatement();
    String dbName = "dlc_db1";
    String createDatabaseSql = String.format("CREATE DATABASE IF NOT
EXISTS %s", dbName);
    statement.execute(createDatabaseSql);
    String tableName = "dlc_t1";
    String wholeTableName = String.format("%s.%s", dbName, tableName);
    String createTableSql =
    String.format(
    "CREATE EXTERNAL TABLE %s (  "
            + "  id string ,   "
            + "  name string ,   "
            + "  status string ,   "
            + "  type string )  "
            + "ROW FORMAT SERDE   "
            + "  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'  "
            + "STORED AS INPUTFORMAT   "
            + "  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'   "
            + "OUTPUTFORMAT   "
            + "  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat' "
            + "LOCATION\\n"
            + "  'cosn://<bucket_name>/<path>'  ",
            wholeTableName);
    statement.execute(createTableSql);
    // get meta data
    DatabaseMetaData metaData = connection.getMetaData();
    System.out.println("product = " +
metaData.getDatabaseProductName());
    System.out.println("jdbc version = "
    + metaData.getDriverMajorVersion() + ", "
    + metaData.getDriverMinorVersion());
    ResultSet tables = metaData.getTables(null, dbName, tableName,
null);
```

```
    while (tables.next()) {
        String name = tables.getString("TABLE_NAME");
        System.out.println("table: " + name);
        ResultSet columns = metaData.getColumns(null, dbName, name,
null);
        while (columns.next()) {
        System.out.println(
        columns.getString("COLUMN_NAME") + "\\t" +
        columns.getString("TYPE_NAME") + "\\t" +
        columns.getInt("DATA_TYPE"));
    }
    columns.close();
    }
    tables.close();
    statement.close();
    connection.close();
}
}
```

Data Query

```
import java.sql.*;
public class DataTest {
    public static void main(String[] args) throws SQLException {
        try {
            Class.forName("com.tencent.cloud.dlc.jdbc.DlcDriver");
            } catch (ClassNotFoundException e) {
    e.printStackTrace();
    return;
    }
    Connection connection = DriverManager.getConnection(
    "jdbc:dlc:<dlc_endpoint>?task_type=<task_type>&database_name=
<database_name>&datasource_connection_name=DataLakeCatalog&region=
<region>&data_engine_name=<data_engine_name>&result_type=<result_type>",
    "<secret_id>",
    "secret_key");
    Statement statement = connection.createStatement();
    String sql = "select * from dlc_test";
```

```
        statement.execute(sql);
        ResultSet rs = statement.getResultSet();
        while (rs.next()) {
            System.out.println(rs.getInt(1) + ":" + rs.getString(2));
            }
            rs.close();
            statement.close();
            connection.close();
            }
    }
```

Database Client

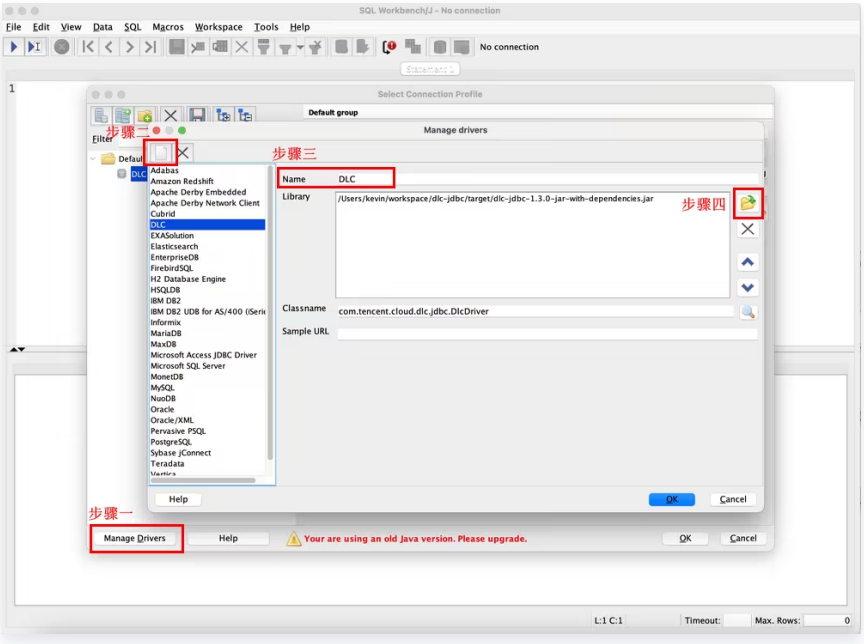
Load the DLC JDBC driver package into your SQL client, connect to the DLC service, and run queries.

Prerequisites

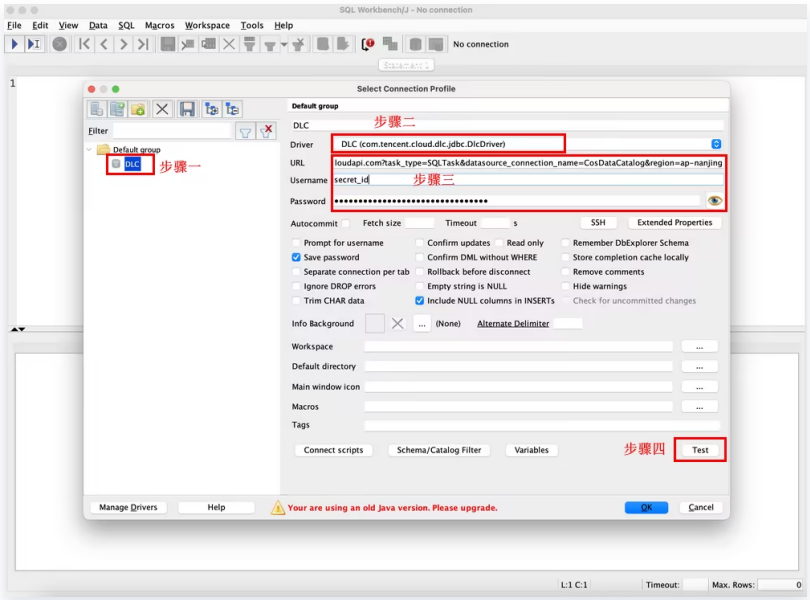
1. The Data Lake Compute service has been activated.
2. The JDBC driver package mentioned above has been downloaded.
3. **SQL Workbench/J** has been downloaded and installed.

Operation Steps

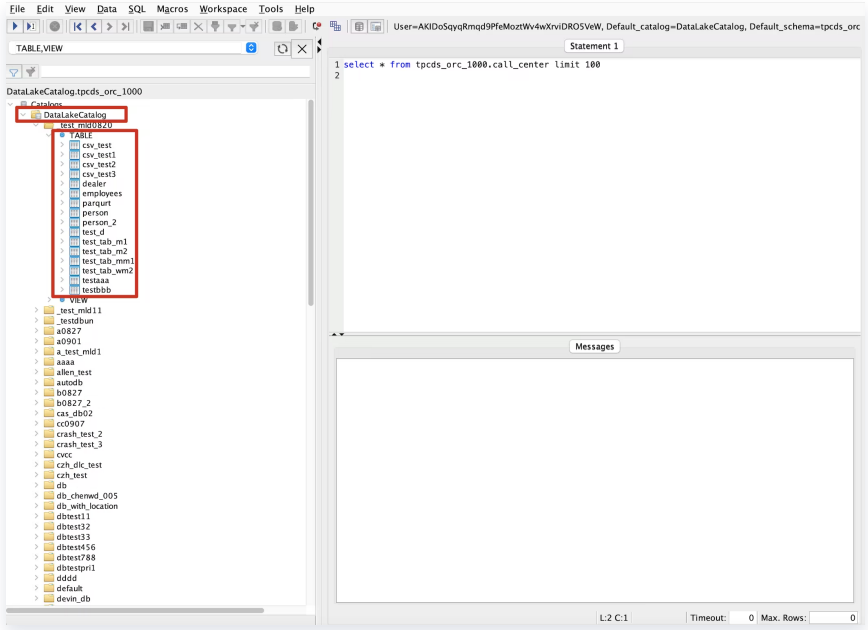
1. Create a DLC Driver using the JDBC driver package.



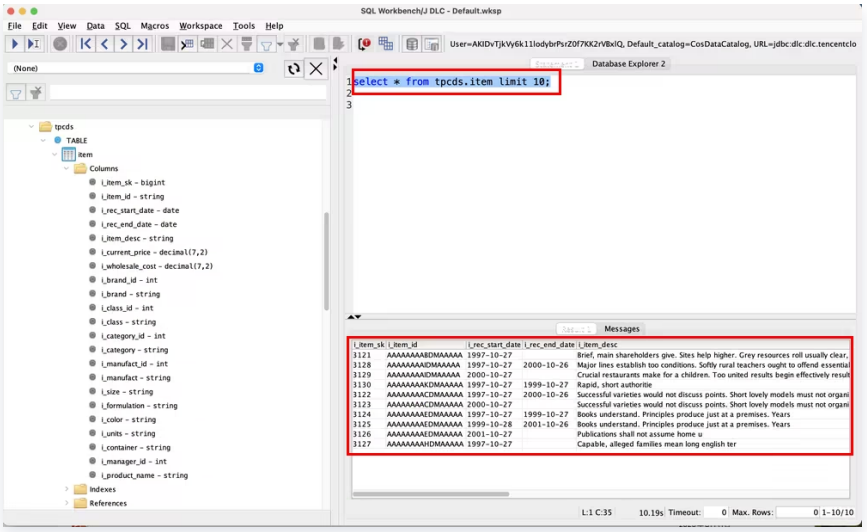
2. Connect to DLC, enter the following parameters, click **test**, and complete the connection to DLC after the test passes.
   - Name: The connection name, which is used to identify the connection to DLC.
   - Username: Corresponds to the secret_id of a Tencent Cloud user.
   - Password: Corresponds to the secret_key of a Tencent Cloud user.
   - URL: The URL used to connect to DLC. Its format is the same as the URL for creating a connection via JDBC mentioned above.



3. View database and table information.



4. Query data.



Hive JDBC Access

Last updated: 2026-05-20 16:42:25

Supported Engine Types

- Standard Spark Engine

Preparing the Environment

- Dependency: JDK 1.8
- JDBC Download: Click to download hive-jdbc-3.1.2-standalone.jar.

Connecting to a Standard Spark Engine

Creating a Service Access Link

Go to the Data Engine page, click the gateway details button, and then go to the gateway details page:

Click **Create Private Link**, select the VPC and subnet where the submission server resides, and then click **Create**. Two access links are generated, one for the Hive2 protocol and the other for the Presto protocol. Use the Hive2 protocol to connect to the Standard Spark Engine, as shown in the following figure.

> ⚠ **Note:**
> Creating a Private Link establishes network connectivity between the engine network and the selected VPC. The submission server can be any accessible server within the selected VPC and is used solely for task submission. If there is no submission server in the selected VPC, you can create a new server to serve as the submission server.

```
jdbc:hive2://{endpoint}:10009/?spark.engine=
{DataEngineName};spark.resourcegroup={ResourceGroupName};secretkey=
{SecretKey};secretid={SecretId};region=
{Region};kyuubi.engine.type=SPARK_SQL;kyuubi.engine.share.level=ENGINE
```

Loading the JDBC Driver

```
Class.forName("org.apache.hive.jdbc.HiveDriver");
```

Creating a Connection via DriverManager

```
jdbc:hive2://{endpoint}:10009/?spark.engine=
{DataEngineName};spark.resourcegroup={ResourceGroupName};secretkey=
{SecretKey};secretid={SecretId};region=
{Region};kyuubi.engine.type=SPARK_SQL;kyuubi.engine.share.level=ENGINE
Properties properties = new Properties();
properties.setProperty("user", {AppId});
Connection cnct = DriverManager.getConnection(url, properties);
```

JDBC Connection String Parameters

Parameter	Required	Description
spark.engine	Yes	Standard Spark Engine Name
spark.resourcegroup	No	Standard Spark engine resource group name. If not specified, temporary resources are created.
secretkey	Yes	The SecretKey in TencentCloud API Key Management
secretid	Yes	The SecretId in TencentCloud API Key Management
region	Yes	Region. Currently, the DLC service supports ap-nanjing, ap-beijing, ap-beijing-fsi, ap-guangzhou, and ap-shanghai. ap-chengdu,ap-chongqing, na-siliconvalley, ap-singapore, ap-hongkong, na-ashburn, eu-frankfurt, ap-shanghai-fsi

kyuubi.engine.type	Yes	Fixed value: SparkSQLTask
kyuubi.engine.share.level	Yes	Fixed value: ENGINE
user	Yes	User AppID.

Complete Data Query Example

```
import org.apache.hive.jdbc.HiveStatement;
import java.sql.*;
import java.util.Properties;

public class TestStandardSpark {
    public static void main(String[] args) throws SQLException {
        try {
            Class.forName("org.apache.hive.jdbc.HiveDriver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            return;
        }
        String url = "jdbc:hive2://{endpoint}:10009/?spark.engine=
{DataEngineName};spark.resourcegroup={ResourceGroupName};secretkey=
{SecretKey};secretid={SecretId};region=
{Region};kyuubi.engine.type=SPARK_SQL;kyuubi.engine.share.level=ENGINE";
        Properties properties = new Properties();
        properties.setProperty("user", {AppId});
        Connection connection = DriverManager.getConnection(url,
properties);
        HiveStatement statement = (HiveStatement)
connection.createStatement();
        String sql = "SELECT * FROM dlc_test LIMIT 100";
        statement.execute(sql);
        ResultSet rs = statement.getResultSet();
        while (rs.next()) {
            System.out.println(rs.getInt(1) + ":" + rs.getString(2));
```

```
            }
            rs.close();
            statement.close();
            connection.close();
    }
    }
```

After compilation is complete, you can submit the jar package to the submission server for execution.

Presto JDBC Access

Last updated: 2026-05-20 16:42:43

> **Note:**
> The Presto engine is deprecated and only available for existing users.

Supported Engine Types

- Standard Presto Engine

Preparing the Environment

- Dependency: JDK 1.8
- JDBC Download: Click to download presto-jdbc-0.284.jar

> **Note:**
> Currently, only the Standard Presto Engine supports access via Presto JDBC. If you are using the SuperSQL engine, see DLC JDBC Access.

Connecting to a Standard Presto Engine

Creating a Service Access Link

Go to the Data Engine page, click the gateway details button, and then go to the gateway details page.

Click **Create Private Link**, select the VPC and subnet where the submission server resides, and then click **Create**. Two access links are generated, one for the Hive2 protocol and the other for the Presto protocol. The Presto protocol is used to connect to the Standard Presto Engine, as shown in the following figure.

> **Note:**
> Creating a Private Link establishes network connectivity between the engine network and the selected VPC. The submission server can be any accessible server within the selected VPC and is used solely for task submission. If there is no submission server in the selected VPC, you can create a new server to serve as the submission server.

```
jdbc:presto://{endpoint}:10999/?sessionProperties=presto.engine:
{DataEngineName};region:{Region};database:{DatabaseName};catalog:
{Catalog}&extraCredentials=secretkey:{SecretKey};secretid:{SecretId}
```

Loading the JDBC Driver

```
Class.forName("com.facebook.presto.jdbc.PrestoDriver");
```

Creating a Connection via DriverManager

```
String url = "jdbc:presto://{endpoint}:10999/?
sessionProperties=presto.engine:{DataEngineName};region:
{Region};database:{DatabaseName};catalog:
{Catalog}&extraCredentials=secretkey:{SecretKey};secretid:{SecretId}";
Properties properties = new Properties();
properties.setProperty("user", {AppId});
Connection connection = DriverManager.getConnection(url, properties);
```

JDBC Connection String Parameters

Parameter	Required	Description
presto.engine	Yes	Standard Presto engine name
database	Yes	Database name.
secretkey	Yes	The SecretKey in TencentCloud API Key Management
secretid	Yes	The SecretId in TencentCloud API Key Management
region	Yes	Region. Currently, the DLC service supports ap-nanjing, ap-beijing, ap-beijing-fsi, ap-guangzhou, ap-shanghai, ap-chengdu, ap-chongqing, na-siliconvalley, ap-singapore, ap-hongkong, na-ashburn, eu-frankfurt, and ap-shanghai-fsi.

catalog	Yes	Data Catalog Name
extraCredentials	Yes	SecretKey: The SecretKey in TencentCloud API Key Management
		SecretId: The SecretId in TencentCloud API Key Management
user	Yes	User AppID.

Complete Data Query Example

```
import com.facebook.presto.jdbc.PrestoStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Properties;

public class TestJDBCKyuubiPresto {
    public static void main(String[] args) throws SQLException {
        try {
            Class.forName("com.facebook.presto.jdbc.PrestoDriver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            return;
        }
        String url = "jdbc:presto://{endpoint}:10999/?
sessionProperties=presto.engine:{DataEngineName};region:
{Region};database:{DatabaseName};catalog:
{Catalog}&extraCredentials=secretkey:{SecretKey};secretid:{SecretId}";
        Properties properties = new Properties();
        properties.setProperty("user", {AppId});
        Connection connection = DriverManager.getConnection(url,
properties);
        PrestoStatement statement = (PrestoStatement)
connection.createStatement();
        String sql = "show catalogs";
        statement.execute(sql);
        ResultSet rs = statement.getResultSet();
        while (rs.next()) {
```

```
            System.out.println(rs.getString(1));
        }
        rs.close();
        statement.close();
        connection.close();
    }
```

After compilation is complete, you can submit the jar package to the submission server for execution.

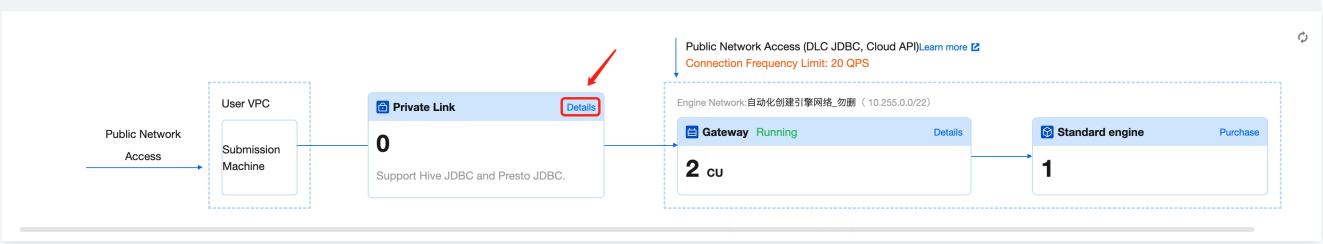
Configuring Public Network Access for a Standard Engine

Last updated: 2026-05-21 16:55:22

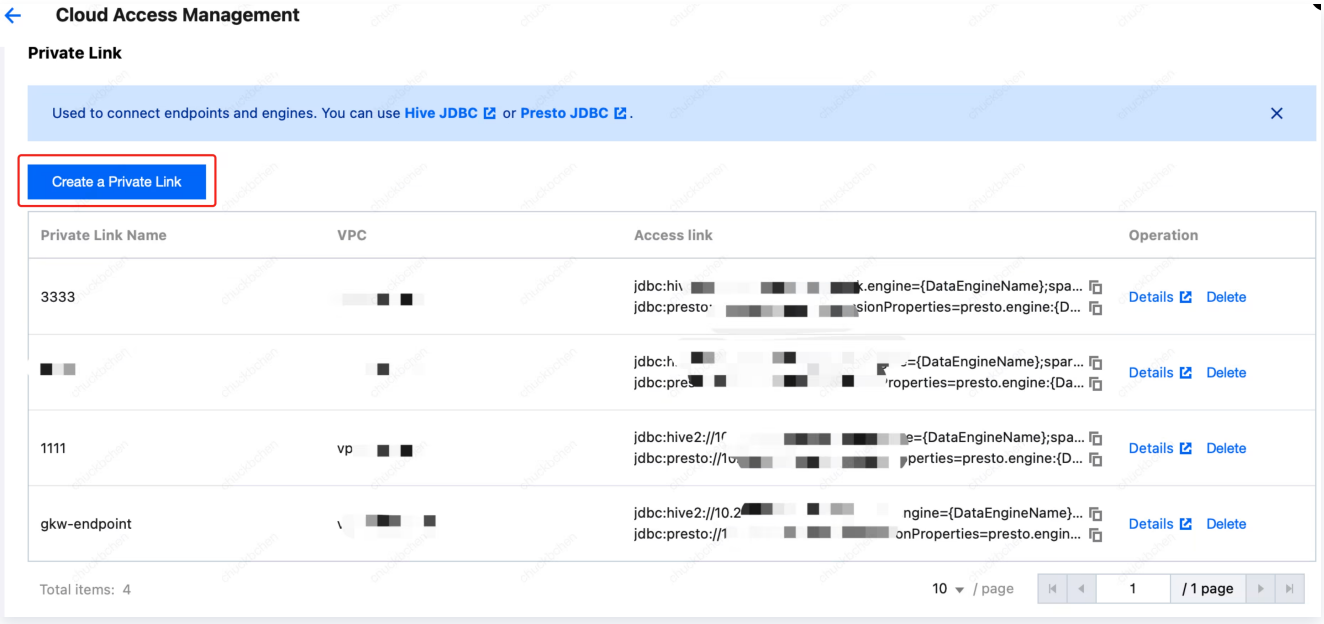
You can access the standard engine from a public network by binding a public IP address to the execution machine.

Step 1: Creating an Endpoint

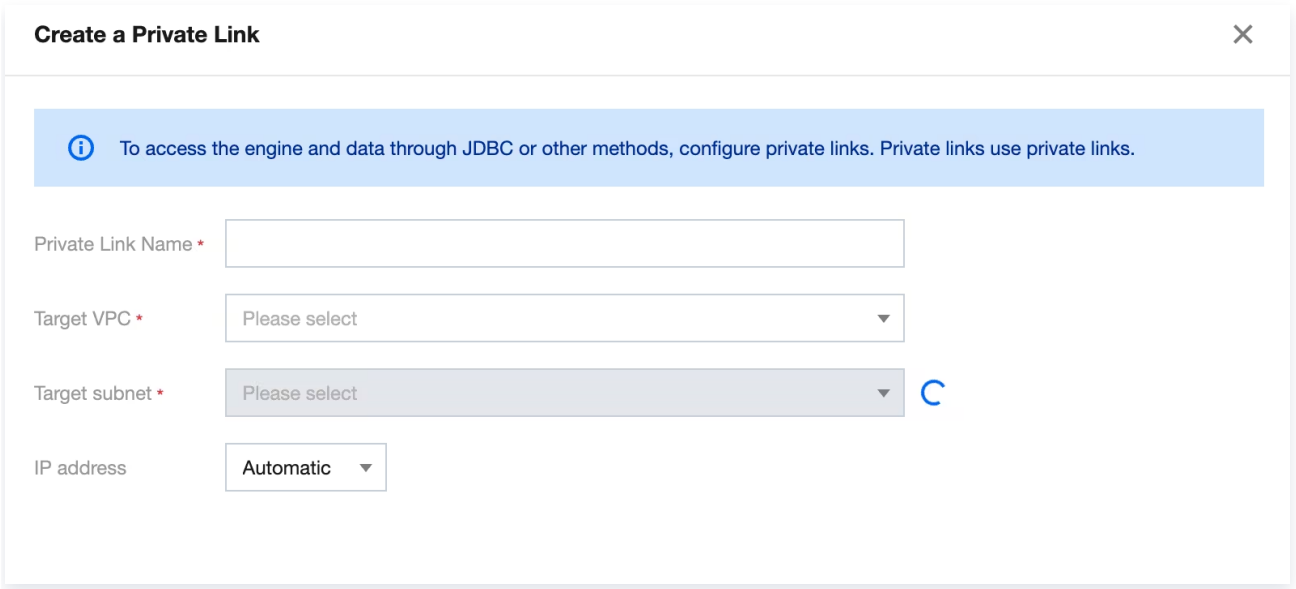
1. Log in to the DLC console, click **Standard Engine** in the left-side menu to go to the Standard Engine page, and then click the Gateway Details button to go to the details page.



On the CAM details page, click the Create Private Link button:



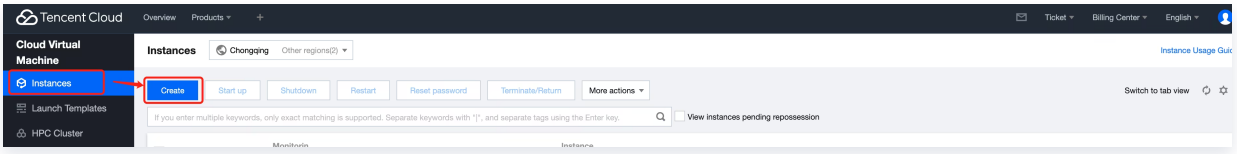
2. Go to the Create Private Link page, select the target VPC and target subnet. The target VPC and subnet you select should be a VPC within your account that is used for communication between users, access points, and the engine.



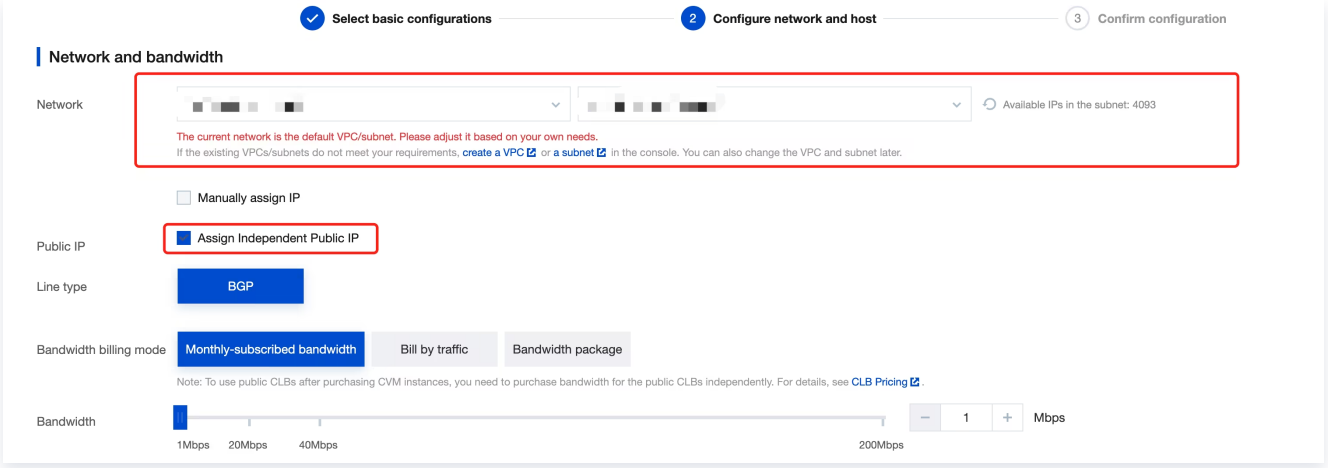
Step 2: Creating an Execution Machine and Enabling Public Network Access

If no execution machine exists, purchase a CVM to serve as the execution machine.

1. Go to the Tencent Cloud CVM console. In the corresponding region, click **Create**.

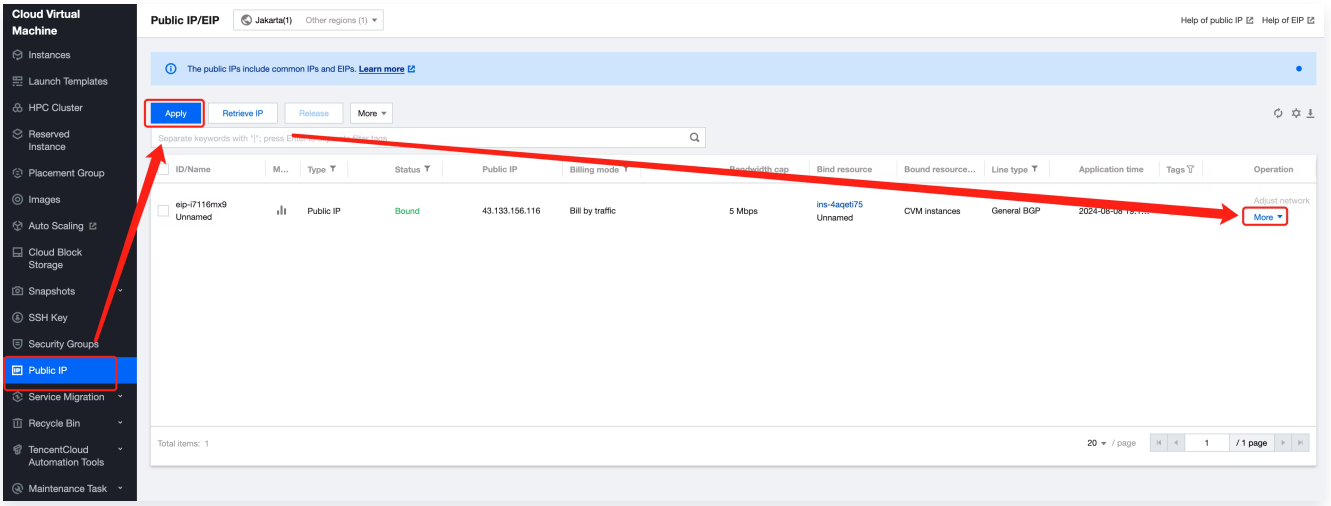


2. The execution machine must be created in a subnet of the VPC whose endpoint was connected in Step 1:
   - Assign a public IP address to the execution machine.
   - The selected security group must have the specific network segments and ports opened.



Step 3: Binding a Public IP Address to the Execution Machine

If an existing CVM is used as the execution machine but does not have an EIP, you can create a new EIP and bind it to the execution machine created in Step 2:



Step 4: Modifying the iptables of the Execution Machine

1. Log in to the execution machine and run the following command on it:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter
iptables -t nat -A PREROUTING -p tcp --dport 10999 -j DNAT --to
$endPointIp:10999
iptables -t nat -A PREROUTING -p tcp --dport 10009 -j DNAT --to
$endPointIp:10009
iptables -t nat -A POSTROUTING -j MASQUERADE
iptables -t nat -L -n -v
```

2. Here, $endPointIP is the private IP address generated after the endpoint is connected on the DLC console, as shown below:

3. After the configuration is complete, you can access the standard engine and submit tasks via the EIP bound to the NAT Gateway created in Step 2.

Note:

- For specific access methods, refer to the documents Hive JDBC Access and Presto JDBC Access.
- You only need to replace the private IP address in the access link with the EIP bound to the NAT Gateway.

Accessing the TDLC CLI

Last updated: 2026-05-20 16:46:54

TDLC is a client-side command-line tool provided by Tencent Cloud DLC (DataLake Compute). Using the TDLC tool, you can submit SQL and Spark jobs to the DLC data engine. TDLC is written in Go and built on the Cobra framework. It supports configuring multiple buckets and performing cross-bucket operations. You can view the usage of TDLC by running `./tdlc [command] --help`.

Download and Installation

The TDLC command-line tool provides binary packages for Windows, macOS, and Linux operating systems. After a simple installation and configuration, it is ready to use. You can select and download the appropriate package based on your client's operating system type.

Operating system	TDLC Binary Package Download URL
Windows	tdlc-windows-v0.2.0
Mac	tdlc-macos-v0.2.0
Linux	tdlc-linux-v0.2.0

Rename the downloaded file to `tdlc`. Open the command line on your client, and navigate to the download directory. If you are using a Mac or Linux system, you need to grant the file executable permissions by running the `chmod +x tdlc` command. After `./tdlc` is executed, if the following content is displayed successfully, the installation is complete and the tool is ready to use.

```
Tencentcloud DLC command tools is used to play around with DLC.
With TDLC user can manger engines, execute SQLs and submit Spark Jobs.

Usage:
  tdlc [flags]
  tdlc [command]

Available Commands:
  config
  help        Help about any command
  spark       Submit spark app to engines.
  sql         Executing SQL.
```

```
  version

Flags:
      --endpoint string          Endpoint of Tencentcloud account.
(default "dlc.tencentcloudapi.com")
      --engine string            DLC engine. (default "public-engine")
  -h, --help                     help for tdlc
      --region string            Region of Tencentcloud account.
      --resource-group string    DLC resource group name of spark
engine.
      --role-arn string          Required by spark jar app.
      --secret-id string         SecretId of Tencentcloud account.
      --secret-key string        SecretKey of Tencentcloud account.
      --token string             Token of Tencentcloud account.

Use "tdlc [command] --help" for more information about a command.
```

Usage Instructions

Global Parameters

TDLC provides the following global parameters.

Global Parameters	Description
--endpoint string	The service connection address, which defaults to dlc.tencentcloudapi.com.
--engine string	The name of the DLC data engine, with a default value of public-engine. It is recommended that you use the dedicated data engine.
--resource-group string	(Optional) The name of the DLC Standard Spark Engine resource group, with a default value of the default resource group default-rg-xxxx.
--region string	Region for use, such as ap-nanjing, ap-beijing, ap-guangzhou, ap-shanghai, ap-chengdu, ap-chongqing, na-siliconvalley, ap-singapore, and ap-hongkong.
--role-arn string	When you submit a Spark job, you need to specify the permissions for accessing COS files. This time, specify the rolearn of the permission role. For details about rolearn, refer to Configure Data Access Policy.

--secret-id string	Tencent Cloud account secretId
--secret-key string	Tencent Cloud account secretKey
--token string	(Optional) Tencent Cloud account temporary token

CONFIG Command

Commonly used parameters can be configured via config, and these configured parameters are provided with default values. Command-line parameters override the parameters already configured in config.

Command	Description
list	List the current default configuration.
set	Changing the configuration
unset	Reset the configuration

Example:

```
./tdlc config list
./tdlc config set secret-id={1} secret-key={2} region={b}
./tdlc config unset region
```

SQL Subcommands

The SQL subcommand currently supports only Presto or SparkSQL clusters. The following are the parameters supported by the SQL subcommand.

Parameter	Description
-e, --exec	Execute SQL statements
-f, --file	Execute SQL files. If there are multiple SQL files, separate them with `;`.
--no-result	Do not obtain results after execution

-p, --progress	Displaying execution progress
-q, --quiet	Quiet mode. Do not wait for the task execution status after the task is submitted.

Example:

```
./tdlc sql -e "SELECT 1" --secret-id aa --secret-key bb --region ap-beijing --engine public-engine --resource-group my-rg
./tdlc sql -f ~/biz.sql --no-result
```

Spark Subcommands

The Spark subcommand includes the following commands, which can be used to submit Spark jobs, view runtime logs, and terminate tasks.

Command	Description
submit	Submit tasks via spark-submit.
run	Running Spark jobs.
log	Viewing Runtime Logs
list	Viewing the Spark job list.
kill	Terminating a Task

The following are the parameters supported by the **Spark submit subcommand**. The file-related parameters listed support the use of local files or the COSN protocol.

Parameter	Description
--driver-size	The driver specification. By default, small, medium, large, and xlarge are used. For memory-optimized clusters, m.small, m.medium, m.large, and m.xlarge are used.
--executor-size	The executor specification. By default, small, medium, large, and xlarge are used. For memory-optimized clusters, m.small, m.medium, m.large, and m.xlarge are used.

--executor-num	executor quantity
--files	Viewing the Spark job list
--archives	Dependent archive files
--class	Main function for Java/Scala execution
--jars	Dependent jar files, separated by `,`
--name	Program name
--py-files	Dependent Python files. The zip, egg, and py formats are supported.
--conf	Additional configurations
--network	Network configuration. The configuration format is: --network "network name"

Example:

```
./tdlc spark submit --name spark-demo1 --engine sparkjar --jars
/root/sparkjar-dep.jar --class com.demo.Example /root/sparkjar-main.jar
arg1
./tdlc spark submit --name spark-demo2 cosn://bucket1/abc.py arg1
```

Third-Party Software Integration

Last updated: 2026-05-20 16:47:11

Tencent Cloud DLC adheres to agile and open principles in its product positioning. It supports extensive integration with third-party software, including scheduling tools, interactive development tools, and BI tools. Currently, it is also continuously supporting and testing integrations with other third-party software.

Note

- For the third-party software listed below, the DLC R&D team has tested the core features of the mainstream editions, but this testing does not cover all version numbers.
- If you encounter issues with integrating DLC with third-party software or have requirements for integrating other third-party software, you are welcome to submit a ticket to contact us.
- If you have requirements for Data Integration and scheduling, you are welcome to use WeData.

Scheduling Tools

If you have already deployed/own the following third-party software, you can connect it to DLC to manage and schedule jobs on DLC.

Third-Party Software	Description	Connection Method
Apache Airflow	Airflow is a workflow management platform that can be used for management, scheduling, and execution.	Use Apache Airflow to schedule the DLC engine for submitting tasks.
Apache DolphinScheduler	DolphinScheduler is a visual DAG workflow task scheduling platform.	Use Apache DolphinScheduler to schedule the DLC engine for submitting tasks.

Python Access

Last updated: 2026-05-20 16:47:55

DLC provides tools that comply with the DBAPI 2.0 standard. You can connect to the Presto/Spark engine of DLC through Python and conveniently operate DLC databases and tables using SQL.

> **Note:**
> The Presto engine is deprecated and only available for existing users.

Preparing the Environment

1. Python 3.9 or later.
2. Install `tencentcloud-dlc-connector`.

```
pip install -i https://mirrors.tencent.com/pypi/simple/ tencentcloud-dlc-connector
```

Usage Examples

Step 1: Connecting to the Engine

Code:

```
import tdlc_connector
import datetime
from tdlc_connector import constants

conn = tdlc_connector.connect(region="<REGION>",
    secret_id="<SECRET_ID>",
    secret_key="<SECRET_KEY>",
    token=None,
    endpoint=None,
    catalog=constants.Catalog.DATALAKECATALOG,
    engine="<ENGINE>",
    resource_group=None,
    engine_type=constants.EngineType.AUTO,
    result_style=constants.ResultStyles.LIST,
    download=False,
```

```
    mode=constants.Mode.LASY,
    database='',
    config={},
    callback=None,
    callback_events=None,
    )
```

Parameter description:

Parameter	Description
region	Region where the engine is located, such as ap-nanjing, ap-beijing, ap-guangzhou, ap-shanghai, ap-chengdu, ap-chongqing, na-siliconvalley, ap-singapore, and ap-hongkong.
secret_id	Tencent Cloud SecretID
secret_key	Tencent Cloud SecretKey
token	(Optional) Temporary key Token
endpoint	(Optional) Service endpoint for connection
engine	Name of the engine in use, for example, "test_python"
resource_group	(Optional) Name of the standard engine resource group. If left empty, a temporary resource group is used. (To use this feature, upgrade the connector to >= 1.2.1.)
engine_type	(Optional) Engine type: corresponds to the engine type of the engine name. For standard engines, you can use this field to specify whether to execute SQL tasks or interactive SQL tasks. Set it to constants.EngineType.SPARK_BATCH to submit interactive SQL, or set it to constants.EngineType.AUTO or others to submit SQL tasks. Default value: constants.EngineType.AUTO. For example: AUTO, PRESTO, SPARK, SPARK_BATCH
result_style	(Optional) Format of the returned result. Options: LIST/DICT.
download	(Optional) Whether to download data directly. Options: True/False. For details, see Download Mode Description.
mode	(Optional) Mode. Supports ALL/LAZY/STREAM.

database	(Optional) Default database
config	(Optional) Configuration for submission to the cluster
callback	(Optional) Callback function with the signature: def cb(statement_id, status)
callback_events	(Optional) Callback trigger events. Used in conjunction with the callback function. For details, see the callback mechanism description.
driver_size	(Optional) Driver node size. Default value: constants.PodSize.SMALL. (Valid only for SPARK_BATCH clusters.) Optional values: SMALL, MEDIUM, LARGE, XLARGE, M_SMALL, M_MEDIUM, M_LARGE, M_XLARGE
executor_size	(Optional) Executor node size. Default value: constants.PodSize.SMALL. (Valid only for SPARK_BATCH clusters.) Optional values: SMALL, MEDIUM, LARGE, XLARGE, M_SMALL, M_MEDIUM, M_LARGE, M_XLARGE
executor_num	(Optional) Number of Executor nodes. Default value: 1. (Valid only for SPARK_BATCH clusters.)
executor_max_num	(Optional) Maximum number of Executor nodes. If this value is not equal to executor_num, dynamic resource allocation is enabled. (Valid only for SPARK_BATCH clusters.)

Download Mode Description:

No.	download	mode	Description
1	False	ALL	Obtain all data from the API. Data can be fetched only after the retrieval is complete.
2	False	LASY	Obtain data from the API. Data is fetched from the server with a delay based on the fetched data volume.
3	False	STREAM	Same as LASY mode
4	True	ALL	Download all results from COS (requires COS read permission). This operation occupies local temporary storage and is recommended for use when the data volume is large.

5	True	LASY	Download results from COS (requires COS read permission). File download is delayed based on the fetched data volume.
6	True	STREAM	Read the result stream from COS in real time (requires COS read permission). Performance is relatively slow, and local memory and disk usage is extremely low.

Step 2: Executing SQL

Code:

```
# Basic Operations

cursor = conn.cursor()
count = cursor.execute("SELECT 1")
print(cursor.fetchone())                    # Reads a row of data.
for row in cursor.fetchall():               # Reads the remaining rows
of data.
    print(row)

# Use the pyformat parameter format

cursor.execute("SELECT * FROM dummy WHERE date < %s",
datetime.datetime.now())
cursor.execute("SELECT * FROM dummy WHERE status in %s", (('SUCCESS',
'INIT', 'FAIL'),))
cursor.execute("SELECT * FROM dummy WHERE date < %(date)s AND status = %
(status)s", {'date': datetime.datetime.now(), 'status': 'SUCCESS'})

# Use the BULK method

cursor.executemany("INSERT INTO dummy VALUES(%s, %s)", [('Zhang San',
18), ('Li Si', 20)])
```

Basic Operations

The code flow above is as follows:

1. A cursor object is created by `conn.cursor()`.
2. An SQL query statement is executed via `cursor.execute("SELECT 1")`, and the result is assigned to the variable `count`.

3. A row of data is read by calling the `cursor.fetchone()` method and then printed.
4. The remaining rows of data are read by calling the `cursor.fetchall()` method within a loop and then printed row by row.

Parameter Passing Methods

Two different parameter passing methods are supported:

- Use the pyformat format: When executing SQL statements, you can use %s as a placeholder and pass the actual parameters in the form of a tuple or dictionary.
- Use the BULK method: Multiple records can be inserted into the database table in batches by calling the `executemany()` method.

Features

Using the Callback Mechanism

```
import tdlc_connector
import datetime
from tdlc_connector import constants


def tdlc_connector_callback(statement_id, state):
    '''
    params: statement_id The task id
    params: state The task status. For the enumeration values, refer to
constants.TaskStatus.
    '''
    print(statement_id, state)


conn = tdlc_connector.connect(region="<REGION>",
    secret_id="<SECRET_ID>",
    secret_key="<SECRET_KEY>",
    engine="<ENGINE>",
    engine_type=constants.EngineType.SPARK,
    result_style=constants.ResultStyles.LIST,
    callback=tdlc_connector_callback,
    callback_events=[constants.CallbackEvent.ON_INIT,
constants.CallbackEvent.ON_SUCCESS]
    )
```

```
cursor = conn.cursor()
cursor.execute("SELECT 1")
cursor.fetchone()

# The callback function is invoked during task initialization and upon
task success.
```

Submitting a Task to a Job Cluster

Task submission to Spark job clusters is now supported. For details, see the following example.

```
from tdlc_connector import constants

conn = tdlc_connector.connect(region="<REGION>",
    secret_id="<SECRET_ID>",
    secret_key="<SECRET_KEY>",
    engine="<ENGINE>",                          # Select the spark job
engine.
    result_style=constants.ResultStyles.LIST,
    driver_size=constants.PodSize.SMALL,        # Select the Driver
specification.
    executor_size=constants.PodSize.SMALL,      # Select the Executor
specification.
    executor_num=1,                             # Set the number of
Executors.
    executor_max_num=1,                         # Set the maximum
number of Executors. If this value is not equal to {executor_num},
dynamic resource allocation is enabled.
)
```

Note:
To use this feature, upgrade the connector to a version greater than or equal to 1.1.0.

Automatic Engine Type Inference

After you specify the engine, you do not need to specify the engine type again. The connector automatically infers it. For details, see the following example.

```
from tdlc_connector import constants

conn = tdlc_connector.connect(region="<REGION>",
    secret_id="<SECRET_ID>",
    secret_key="<SECRET_KEY>",
    engine="<ENGINE>",
    engine_type=constants.EngineType.AUTO      # Set this to AUTO or
omit this parameter. The driver automatically infers the engine type.
)
```

Note:
To use this feature, upgrade the connector to a version greater than or equal to 1.1.0.

Null Value Conversion

The current result set is stored in CSV format. By default, the engine converts null values to empty strings. To distinguish null values, specify a null value indicator, for example, "\1". The engine query result then converts null values to "\1". Meanwhile, the driver converts the "\1" field to **None**. For details, see the following example.

```
from tdlc_connector import constants, formats

formats.FORMAT_STRING_NULL = '\1'

conn = tdlc_connector.connect(region="<REGION>",
    secret_id="<SECRET_ID>",
    secret_key="<SECRET_KEY>",
    engine="<ENGINE>",
    result_style=constants.ResultStyles.LIST
)
```

Note:
Null value conversion is currently supported only on SparkSQL clusters. Upgrade the connector to a version greater than or equal to 1.1.3.