Hive Metastore Client Documentation

Release 0.0.1

QuintoAndar

Aug 29, 2022

CONTENTS

Made with by the **Data Engineering** team from QuintoAndar.

A client for connecting and running DDLs on Hive Metastore using Thrift protocol.

An example of how to use the library for running commands in hive metastore:

```
from hive_metastore_client.builders import DatabaseBuilder
from hive_metastore_client import HiveMetastoreClient

database = DatabaseBuilder(name='new_db').build()
with HiveMetastoreClient(HIVE_HOST, HIVE_PORT) as hive_metastore_client:
    hive_metastore_client.create_database(database)
```

To learn more use cases in practice, see Hive Metastore Client's examples

CHAPTER

ONE

NAVIGATION

1.1 Getting Started

Hive Metastore Client depends on **Python 3.7+**.

Python Package Index hosts reference to a pip-installable module of this library, using it is as straightforward as including it on your project's requirements.

```
pip install hive-metastore-client
```

Or after listing `hive-metastore-client` in your `requirements.txt` file:

```
pip install -r requirements.txt
```

1.1.1 Discovering Hive Metastore Client

Click on the following links to open the examples:

#1 Create a database

#2 Create a table

#3 Create an external table

#4 Add columns to a table

#5 Add partitions to a table if not exists

#6 Add partitions to a table

#7 Get partition keys objects from a table

#8 Get partition keys (names & types) from a table

#9 Get partition keys (names only) from a table

#10 Bulk drop partitions values from a table

#11 Get partition values from a table

1.1.2 Available methods

You can see all the Hive Metastore server available methods by looking at the interface: `thrift_files.libraries.thrift_hive_metastore_client.ThriftHiveMetastore.Iface`.

Beyond the default methods, this library also implements the methods below in the `HiveMetastoreClient` class:

- `add_columns_to_table`
- `drop_columns_from_table`
- `add_partitions_if_not_exists`
- `add_partitions_to_table`
- `create_database_if_not_exists`
- `create_external_table`
- `get_partition_keys_objects`
- `get_partition_keys`
- `get_partition_keys_names`
- `bulk_drop_partitions`
- `get_partition_values_from_table`

1.2 About Thrift

This project uses the Thrift mapping files and the Thrift package to auto generate the python libraries that you can find in `thrift_files/libraries/`.

1.2.1 Updating the Thrift Hive Metastore client

Requirements

The thrift service mapping

Download the *hive_metastore.thrift* (and its required thrift files if any. Then place them in the same directory).

These two files are the mapping for the Thrift Service that the Hive Metastore is built on:

- hive_metastore.thrift
- fb303.thrift

We are using the Hive Metastore version 3.0 (branch-3.0)

Thrift package

1 - First you need to install the thrift package so you can "compile" the thrift file into python code. You can refer to these tutorial for installing it: https://thrift-tutorial.readthedocs.io/en/latest/installation.html. We used the latest version (0.14.0) of the Thrift package.

2 - After installing thrift package, you should open the directory where the *hive_metastore.thrift* (and others) is placed and run:

```
thrift --gen py hive_metastore.thrift
thrift --gen py fb303.thrift
```

3 - Now you have the new python code generated inside `gen-py`. Extract the classes and place them in the right directories - we use `thrift_files/libraries/` for these generate libraries.

4 - The generated files are huge, therefore be sure that the generated files directory names are ignored in the make commands *style-check* and *apply-lint*. So these files are not evaluated during the checks.

1.3 Builders

In order to isolate its dependencies from users code, this library implements the builder pattern.

So instead of manually creating the Thrift objects for dealing with the Hive Metastore server, you can use the builders.

Each builder reflects the Thrift object's characteristics that it is building. For more information about the object particularities, you can try finding more information in the hive_metastore.thrift mapping.

You should always call the `.build()` at the end to get the desired object.

Some objects (like the Thrift `Table`) are more complex and require other objects as parameters.

To learn more in practice use cases, see Hive Metastore Client examples.

1.4 API Specification

1.4.1 hive_metastore_client package

Subpackages

hive_metastore_client.builders package

Submodules

Abstract Builder Class.

class hive_metastore_client.builders.abstract_builder.**AbstractBuilder**

Bases: `object`

Abstract Builder class with builder methods.

abstract build() → object

Builds the respective Thrift object.

Returns
an instance of the built object

ColumnBuilder.

class hive_metastore_client.builders.column_builder.**ColumnBuilder**(*name: str*, *type: str*, *comment: Optional[str] = None*)

Bases: *AbstractBuilder*

Builds thrift FieldSchema object.

build() → FieldSchema

Returns the thrift FieldSchema object.

DatabaseBuilder.

class hive_metastore_client.builders.database_builder.**DatabaseBuilder**(*name: str*, *description: Optional[str] = None*, *location_uri: Optional[str] = None*, *parameters: Optional[Mapping[str, str]] = None*, *privileges: Optional[PrincipalPrivilegeSet] = None*, *owner_name: Optional[str] = None*, *owner_type: Optional[PrincipalType] = None*, *catalog_name: Optional[str] = None*)

Bases: *AbstractBuilder*

Builds thrift Database object.

build() → Database

Returns the thrift Database object.

PartitionBuilder.

class hive_metastore_client.builders.partition_builder.**PartitionBuilder**(*values: List[str]*, *db_name: str*, *table_name: str*, *sd: Optional[StorageDescriptor] = None*, *create_time: Optional[int] = None*, *last_access_time: Optional[int] = None*, *parameters: Optional[Mapping[str, str]] = None*, *privileges: Optional[PrincipalPrivilegeSet] = None*, *cat_name: Optional[str] = None*)

Bases: object

Builds thrift Partition object.

build() → Partition

Returns the thrift Partition object.

SerDeInfoBuilder.

class hive_metastore_client.builders.serde_info_builder.**SerDeInfoBuilder**(*name: Optional[str] = None*, *serialization_lib: Optional[str] = None*, *parameters: Optional[Dict[str, str]] = None*, *description: Optional[str] = None*, *serializer_class: Optional[str] = None*, *deserializer_class: Optional[str] = None*, *serde_type: Optional[SerdeType] = None*)

Bases: *AbstractBuilder*

Builds thrift table's Serialization-Deserialization info object.

build() → SerDeInfo

Returns the thrift SerDeInfo object.

StorageDescriptorBuilder.

class hive_metastore_client.builders.storage_descriptor_builder.**StorageDescriptorBuilder**(*columns: List[FieldSchema] = None*, *location: str = None*, *input_format: str = None*, *output_format: str = None*, *serde_info: SerDeInfo = None*, *compressed: Optional[bool] = None*, *num_buckets: Optional[int] = None*, *bucket_cols: Optional[List[str]] = None*, *sort_cols: Optional[List[Order]] = None*, *parameters: Optional[Dict[str, str]] = None*, *skewed_info: Optional[SkewedInfo] = None*, *stored_as_sub_dirs: Optional[bool] = None*)

Bases: *AbstractBuilder*

Builds thrift StorageDescriptor object.

build() → StorageDescriptor

Returns the thrift StorageDescriptor object.

TableBuilder.

class hive_metastore_client.builders.table_builder.**TableBuilder**(*table_name: str*, *db_name: str*, *storage_descriptor: StorageDescriptor*, *owner: Optional[str] = None*, *create_time: Optional[int] = None*, *last_access_time: Optional[int] = None*, *retention: Optional[int] = None*, *partition_keys: Optional[List[FieldSchema]] = None*, *parameters: Optional[Dict[str, str]] = None*, *view_original_text: Optional[str] = None*, *view_expanded_text: Optional[str] = None*, *table_type: Optional[str] = None*, *privileges: Optional[PrincipalPrivilegeSet] = None*, *temporary: bool = False*, *rewrite_enabled: Optional[bool] = None*, *creation_metadata: Optional[CreationMetadata] = None*, *cat_name: Optional[str] = None*, *owner_type: PrincipalType = 1*)

Bases: *AbstractBuilder*

Builds thrift Table object.

build() → Table

Returns the thrift Table object.

Module contents

Builders for helping library users to create the Thrift objects.

Submodules

Hive Metastore Client main class.

class hive_metastore_client.hive_metastore_client.**HiveMetastoreClient**(*host: str*, *port: int = 9083*)

Bases: `Client`

User main interface with the metastore server methods.

COL_TYPE_INCOMPATIBILITY_DISALLOW_CONFIG = 'hive.metastore.disallow.incompatible.col.type.changes'

add_columns_to_table(*db_name: str*, *table_name: str*, *columns: List[FieldSchema]*) → None

Adds columns to a table.

Parameters

- **db_name** – database name of the table
- **table_name** – table name
- **columns** – columns to be added to the table

add_partitions_if_not_exists(*db_name: str*, *table_name: str*, *partition_list: List[Partition]*) → None

Add partitions to a table if it does not exist.

If a partition is added twice, the method handles the
AlreadyExistsException, not raising the exception.

Parameters

- **db_name** – database name where the table is at
- **table_name** – table name which the partitions belong to
- **partition_list** – list of partitions to be added to the table

add_partitions_to_table(*db_name: str*, *table_name: str*, *partition_list: List[Partition]*) → None

Add partitions to a table.

If any partition of partition_list already exists, an
AlreadyExistsException, will be thrown and no partition will be added.

Parameters

- **db_name** – database name where the table is at
- **table_name** – table name which the partitions belong to
- **partition_list** – list of partitions to be added to the table

bulk_drop_partitions(*db_name: str*, *table_name: str*, *partition_list: List[List[str]]*, *delete_data: bool = False*) → None

Drops the partitions values from the partition list.

This methods simulates a bulk drop for the user, since the server only supports an unitary drop.

If some partition cannot be dropped an exception will be thrown in the end of execution.

Parameters

- **db_name** – database name of the table
- **table_name** – table name
- **partition_list** – the partitions to be dropped
- **delete_data** – indicates whether the data respective to the partition should be dropped in the source.

Raises
NoSuchObjectException

close() → None

Closes the connection with the Thrift server.

create_database_if_not_exists(*database: Database*) → None

Creates the table in Hive Metastore if it does not exist.

Since hive metastore server and thrift mapping do not have the option of checking if the database does not exist, this method simulates this this behavior.

Parameters
database – the database object

create_external_table(*table: Table*) → None

Creates an external table in Hive Metastore.

When a table is created with tableType default (None) or equal to EXTERNAL_TABLE there is an error in the server that creates the table as a MANAGED_TABLE.

This method enforces the parameter EXTERNAL=TRUE so the table is created correctly.

Parameters
table – the table object

drop_columns_from_table(*db_name: str*, *table_name: str*, *columns: List[str]*) → None

Drops columns from a table.

It encapsulates the logic of calling alter table with removed columns from the list of columns, since hive does not have a drop command.

Parameters

- **db_name** – database name of the table
- **table_name** – table name
- **columns** – names of the columns to be dropped from the table

get_partition_keys(*db_name: str*, *table_name: str*) → List[Tuple[str, str]]

Gets the partition keys from a table as a tuple: (name, type).

An empty list will be returned when no table is found or when the table has no partitions.

Parameters

- **db_name** – database name where the table is at
- **table_name** – table name which the partition keys belong to

get_partition_keys_names(*db_name: str*, *table_name: str*) → List[str]

Gets the partition keys names from a table.

An empty list will be returned when no table is found or when the table has no partitions

Parameters

- **db_name** – database name where the table is at
- **table_name** – table name which the partition keys belong to

get_partition_keys_objects(*db_name: str*, *table_name: str*) → List[FieldSchema]

Gets the partition keys objects, containing the metadata, from a table.

An empty list will be returned when no table is found or when the table has no partitions

Parameters

- **db_name** – database name where the table is at
- **table_name** – table name which the partition keys belong to

get_partition_values_from_table(*db_name: str*, *table_name: str*) → List[List[str]]

Gets the partition names from a table.

It automatically fetches the table's partition keys.

An empty list will be returned when no table is found or when the table has no partitions.

Parameters

- **db_name** – database name where the table is at
- **table_name** – table name which the partitions belong to

open() → *HiveMetastoreClient*

Opens the connection with the Thrift server.

Returns
HiveMetastoreClientConnector instance

Module contents

Hive Metastore Client.

PYTHON MODULE INDEX

h

INDEX

M

O

P

S

T