sfu

Release 2.0.0

Nth Party, Ltd.

Feb 11, 2022

CONTENTS

Snowflake URI utility library that supports extraction of Snowflake configuration data and method parameters from Snowflake resource URIs.

CHAPTER

ONE

PURPOSE

When applications that employ the Snowflake Python SDK must work with resources that are spread across multiple accounts, it can be useful to tie Snowflake configuration information (both credentials and resource data) directly tot associated Snowflake resources (*e.g.*, by including the configuration data within URIs). This library provides a class that extracts Snowflake configuration data and method parameters from a URI, offering a succint syntax for passing (directly into Snowflake methods) configuration data and/or resource names that are included within URIs.

CHAPTER

TWO

PACKAGE INSTALLATION AND USAGE

The package is available on PyPI:

```
python -m pip install sfu
```

The sfu class can be imported with:

```
from sfu import sfu
```

The class provides methods for extracting configuration data (credentials and non-credentials) from URIs, as in the examples below:

```
>>> from sfu import sfu
>>> import snowflake.connector

# Create a connector client given a URI (for a table in some snowflake database) that
# includes credentials (a username 'ABC', a password 'XYZ', and an associated account
# 'UVW').
# Make sure the account contains the region and platform, e.g., xxx.us-east-1.aws.
>>> s = sfu("snow://ABC:XYZ:UVW@DATABASE")
>>> conn = connector.connect(**s.credentials())

# It can also be useful to bind a connection to some database and some data processing
# warehouse, so you don't need to execute cursor commands later. The following will
# return a connector client that is configured against DATABASE, using WH for data
# processing.
>>> uri = "snow://ABC:XYZ:UVW@DATABASE/TABLE@warehouse=WH"
>>> s = sfu(uri)
>>> c = connector.connect(**s.for_connection())
>>> cs = c.cursor()
>>> cs.execute(f"SELECT col1,col2 FROM {s.for_table()}")

# Note that this is equivalent to the following:
>>> s = sfu(uri)
>>> c = connector.connect(**s.credentials())
>>> cs = c.cursor()
>>> cs.execute(f"USE DATABASE {s.for_db()}")
>>> cs.execute(f"USE WAREHOUSE {s.for_warehouse()}")
>>> cs.execute(f"SELECT col1,col2 FROM {s.for_table()}")
```

CHAPTER

THREE

DEVELOPER NOTES

Pipenv is used for dependency management of the main library, minus Read the Docs which does not support Pipenv. You can install all dependencies with:

```
pipenv install --dev
```

To release a new version of the library, run:

```
pipenv run python -m pip install --upgrade build twine
pipenv run python -m build
pipenv run python -m twine upload dist/*
```

CHAPTER

FOUR

DOCUMENTATION

The documentation can be generated automatically from the source files using Sphinx:

```
python -m pip install -e .
cd docs
python -m pip install -r requirements.txt
sphinx-apidoc -f -E --templatedir=_templates -o _source .. && make html
```

CHAPTER

FIVE

TESTING AND CONVENTIONS

All unit tests are executed and their coverage is measured when using pytest:

```
pipenv run python -m pytest --cov=sfu --cov-report term-missing
```

Style conventions are enforced using Pylint:

```
pipenv run python -m flake8 src/sfu
```

CHAPTER

SIX

CONTRIBUTIONS

In order to contribute to the source code, open an issue or submit a pull request on the GitHub page for this library.

CHAPTER

SEVEN

VERSIONING

The version number format for this library and the changes to the library associated with version number increments conform with Semantic Versioning 2.0.0.