Tentaclio

octoenergy

Apr 20, 2023

CONTENTS:

CHAPTER

ONE

TENTACLIO

Python library that simplifies:

- Handling streams from different protocols such as `file:`, `ftp:`, `sftp:`, `s3:`, ...
- Opening database connections.
- Managing the credentials in distributed systems.

Main considerations in the design:

- Easy to use: all streams are open via `tentaclio.open`, all database connections through `tentaclio.db`.
- URLs are the basic resource locator and db connection string.
- Automagic authentication for protected resources.
- Extensible: you can add your own handlers for other schemes.
- Pandas interaction.

CHAPTER

TWO

QUICK EXAMPLES.

2.1 Read and write streams.

```
import tentaclio
contents = " "

with tentaclio.open("ftp://localhost:2021/upload/file.txt", mode="w") as writer:
    writer.write(contents)

# Using boto3 authentication under the hood.
bucket = "s3://my-bucket/octopus/hello.txt"
with tentaclio.open(bucket) as reader:
    print(reader.read())
```

2.2 Copy streams

```
import tentaclio

tentaclio.copy("/home/constantine/data.csv", "sftp://constantine:tentacl3@sftp.
↪octoenergy.com/uploads/data.csv")
```

2.3 Delete resources

```
import tentaclio

tentaclio.remove("s3://my-bucket/octopus/the-9th-tentacle.txt")
```

2.4 List resources

```
import tentaclio

for entry in tentaclio.listdir("s3:://mybucket/path/to/dir"):
    print("Entry", entry)
```

2.5 Authenticated resources.

```
import os

import tentaclio

print("env ftp credentials", os.getenv("OCTOIO__CONN__OCTOENERGY_FTP"))
# This prints `sftp://constantine:tentacl3@sftp.octoenergy.com/`

# Credentials get automatically injected.

with tentaclio.open("sftp://sftp.octoenergy.com/uploads/data.csv") as reader:
    print(reader.read())
```

2.6 Database connections.

```
import os

import tentaclio

print("env TENTACLIO__CONN__DB", os.getenv("TENTACLIO__CONN__DB"))

# This prints `postgresql://octopus:tentacle@localhost:5444/example`

# hostname is a wildcard, the credentials get injected.
with tentaclio.db("postgresql://hostname/example") as pg:
    results = pg.query("select * from my_table")
```

2.7 Pandas interaction.

```
import pandas as pd  #
import tentaclio  #

df = pd.DataFrame([[1, 2, 3], [10, 20, 30]], columns=["col_1", "col_2", "col_3"])

bucket = "s3://my-bucket/data/pandas.csv"

with tentaclio.open(bucket, mode="w") as writer:  # supports more pandas readers
```

(continues on next page)

(continued from previous page)

```
    df.to_csv(writer, index=False)

with tentaclio.open(bucket) as reader:
    new_df = pd.read_csv(reader)

# another example: using pandas.DataFrame.to_sql() with tentaclio to upload
with tentaclio.db(
        connection_info,
        connect_args={'options': '-csearch_path=schema_name'}
    ) as client:
    df.to_sql(
        name='observations', # table name
        con=client.conn,
    )
```

CHAPTER

THREE

INSTALLATION

You can get tentaclio using pip

```
pip install tentaclio
```

or pipenv

```
pipenv install tentaclio
```

3.1 Developing.

Clone this repo and install pipenv:

In the `Makefile` you'll find some useful targets for linting, testing, etc. i.e.:

```
make test
```

3.2 How to use

This is how to use `tentaclio` for your daily data ingestion and storing needs.

3.2.1 Streams

In order to open streams to load or store data the universal function is:

```
import tentaclio

with tentaclio.open("/path/to/my/file") as reader:
    contents = reader.read()

with tentaclio.open("s3://bucket/file", mode='w') as writer:
    writer.write(contents)
```

Allowed modes are `r`, `w`, `rb`, and `wb`. You can use `t` instead of `b` to indicate text streams, but that's the default.

In order to keep tentaclio as light as possible, it only includes `file`, `ftp`, `sftp`, `http` and `https` schemes by default. However, many more are easily available by installing extra packages:

Default:

- `/local/file`
- `file:///local/file`
- `ftp://path/to/file`
- `sftp://path/to/file`
- `http://host.com/path/to/resource`
- `https://host.com/path/to/resource`

tentaclio-s3

- `s3://bucket/file`

tentaclio-gs

- `gs://bucket/file`
- `gsc://bucket/file`

tentaclio-gdrive

- `gdrive:/My Drive/file`
- `googledrive:/My Drive/file`

tentaclio-postgres

- `postgresql://host/database::table` will allow you to write from a csv format into a database with the same column names (note that the table goes after `::` :warning:).

You can add the credentials for any of the urls in order to access protected resources.

You can use these readers and writers with pandas functions like:

```
import pandas as pd
import tentaclio

with tentaclio.open("/path/to/my/file") as reader:
    df = pd.read_csv(reader)

[...]

with tentaclio.open("s3::/path/to/my/file", mode='w') as writer:
    df.to_parquet(writer)
```

`Readers`, `Writers` and their closeable versions can be used anywhere expecting a file-like object; pandas or pickle are examples of such functions.

Notes on writing files for Spark, Presto, and similar downstream systems

The default behaviour for the `open` context manager in python is to create an empty file when opening it in writable mode. This can be annoying if the process that creates the data within the `with` clause yields empty dataframes and nothing gets written. This will make Spark and Presto panic.

To avoid this we can make the stream *empty safe* so the empty buffer won't be flushed if no writes have been performed so no empty file will be created.

```
with tio.make_empty_safe(tio.open("s3://bucket/file.parquet", mode="wb")) as writer:
    if not df.empty:
        df.to_parquet(writer)
```

3.2.2 File system like operations to resources

Listing resources

Some URL schemes allow listing resources in a pythonnic way:

```
import tentaclio

for entry in tentaclio.listdir("s3:://mybucket/path/to/dir"):
    print("Entry", entry)
```

Whereas `listdir` might be convinient we also offer `scandir`, which returns a list of DirEntrys, and, `walk`. All functions follow as closely as possible their standard library definitions.

3.2.3 Database access

In order to open db connections you can use `tentaclio.db` and have instant access to postgres, sqlite, athena and mssql.

```
import tentaclio

[...]

query = "select 1";
with tentaclio.db(POSTGRES_TEST_URL) as client:
    result =client.query(query)
[...]
```

The supported db schemes are:

Default:

- `sqlite://`
- `mssql://`
- – Any other scheme supported by sqlalchemy.

tentaclio-postgres

- `postgresql://`

tentaclio-athena

- `awsathena+rest://`

tentaclio-databricks

- `databricks+thrift://`

tentaclio-snowflake

- `snowflake://`

Extras for databases

For postgres you can set the variable `TENTACLIO__PG_APPLICATION_NAME` and the value will be injected when connecting to the database.

3.2.4 Automatic credentials injection

1. Configure credentials by using environmental variables prefixed with `TENTACLIO__CONN__` (i.e. `TENTACLIO__CONN__DATA_FTP=sfpt://real_user:132ldsf@ftp.octoenergy.com`).
2. Open a stream:

```
with tentaclio.open("sftp://ftp.octoenergy.com/file.csv") as reader:
    reader.read()
```

The credentials get injected into the url.

3. Open a db client:

```
import tentaclio

with tentaclio.db("postgresql://hostname/my_data_base") as client:
    client.query("select 1")
```

Note that `hostname` in the url to be authenticated is a wildcard that will match any hostname. So `authenticate("http://hostname/file.txt")` will be injected to `http://user:pass@octo.co/file.txt` if the credential for `http://user:pass@octo.co/` exists.

Different components of the URL are set differently:

- Scheme and path will be set from the URL, and null if missing.
- Username, password and hostname will be set from the stored credentials.
- Port will be set from the stored credentials if it exists, otherwise from the URL.
- Query will be set from the URL if it exists, otherwise from the stored credentials (so it can be overriden)

Credentials file

You can also set a credentials file that looks like:

```
secrets:
    db_1: postgresql://user1:pass1@myhost.com/database_1
    db_2: mssql://user2:pass2@otherhost.com/database_2?
↪driver=ODBC+Driver+17+for+SQL+Server
    ftp_server: ftp://fuser:fpass@ftp.myhost.com
```

And make it accessible to tentaclio by setting the environmental variable `TENTACLIO__SECRETS_FILE`. The actual name of each url is for traceability and has no effect in the functionality.

(Note that you may need to add `?driver={driver from /usr/local/etc/odbcinst.ini}` for mssql database connection strings; see above example)

Alternatively you can run `curl https://raw.githubusercontent.com/octoenergy/tentaclio/master/extras/init_tentaclio.sh` to create a secrets file in `~/.tentaclio.yml` and automatically configure your environment.

3.3 Quick note on protocols structural subtyping.

In order to abstract concrete dependencies from the implementation of data related functions (or in any part of the system really) we use typed protocols. This allows a more flexible dependency injection than using subclassing or more complex approches. This idea is heavily inspired by how this exact thing is done in go. Learn more about this principle in our tech blog.

CHAPTER

FOUR

DOCUMENTATION

Tentaclio uses reStructuredText (Restructured Text), MyST (Markedly Structured Text) and the Sphinx documentation system. This allows it to be built into other forms for easier viewing and browsing.

4.1 Building docs

To create an HTML version of the docs, use:

```
$ cd docs

$ make html
```

This will generate a static set of HTML files with root *docs/_build/html/index.html*. This can be viewed in a browser

CHAPTER

FIVE

INDICES AND TABLES

- genindex