pyfeder8

Release 2.4

Peter Moorthamer

Jul 19, 2023

CONTENTS:

CHAPTER

ONE

ABOUT THIS PROJECT

pyfeder8 is a Python library to interact with the Feder8 central and local services

Useful links:

- Python package: pyfeder8
- Source code: pyfeder8 source
- Central Catalogue Service: API documentation
- **Projects:**
  - HONEUR Portal
  - ATHENA Portal
  - ESFURN Portal
  - LupusNet Portal
  - PHederation Portal

CHAPTER

TWO

CODE DOCS

2.1 CatalogueClient

class pyfeder8.catalogue.CatalogueClient.**CatalogueClient**(*configuration: Configuration*)

A Python client to call the REST API of the Feder8 Catalogue Service The configuration determines the domain and environment to connect to

download_file_with_key(*file_key: str*, *token_context:* TokenContext | *None = None*)

Downloads the file with the given file key

download_file_with_key_as_dataframe(*file_key: str*, *token_context:* TokenContext | *None = None*)

Downloads the file with the given file key and returns it as pandas DataFrame

download_file_with_key_as_dictionary(*file_key: str*, *token_context:* TokenContext | *None = None*)

Downloads the file with the given file key and returns it as dictionary

get_study(*study_id: int*, *token_context:* TokenContext | *None = None*)

Retrieves the study with the given ID from the Study Catalogue

list_files(*file_key_prefix: str*, *token_context:* TokenContext | *None = None*)

Returns a list of files whose file key matches the given file key prefix

list_script_results(*study_id: int*, *script_uuid=None*, *script_version_uuid=None*, *latest_version_only=False*, *token_context:* TokenContext | *None = None*)

Retrieves a list of script results for the given study

list_studies(*token_context:* TokenContext | *None = None*)

Retrieves a list of all studies from the Study Catalogue

save_dataframe_as_csv_script_result(*script_version_uuid*, *df: DataFrame*, *filename: str*, *token_context:* TokenContext | *None = None*)

Saves the given data frame as CSV file for the script with the given UUID

save_dataframe_as_json_script_result(*script_version_uuid*, *df: DataFrame*, *filename: str*, *token_context:* TokenContext | *None = None*)

Saves the given data frame as JSON file for the script with the given UUID

save_dictionary_as_json_script_result(*script_version_uuid*, *dd: dict*, *filename: str*, *token_context:* TokenContext | *None = None*)

Saves the given data dictionary as JSON file for the script with the given UUID

save_script_result(*script_version_uuid*, *result_file*, *token_context:* TokenContext | *None = None*)

Saves the given result for the script with the given UUID

2.2 ConfigurationClient

class pyfeder8.config.ConfigurationClient.**ConfigurationClient**(*config_server='http://config-server:8080/config-server'*, *config_name='feder8-config'*, *username='root'*, *password='s3cr3t'*)

Client to retrieve a configuration from a local config server

2.3 DistributedAnalyticsClient

class pyfeder8.distributed_analytics.DistributedAnalyticsClient.**DistributedAnalyticsClient**(*configuration: Configuration*)

A Python client to call the REST API of the Feder8 Distributed Analytics Service The configuration determines the domain and environment to connect to

2.4 DatabaseConnection

pyfeder8.DatabaseConnection.**get_db_engine**(*db_conn_details: DatabaseConnectionDetails*)

Helper function to create a database engine by use of the given connection parameters

2.5 Feder8Service

class pyfeder8.Feder8Service.**Feder8Service**(*configuration: Configuration*)

A helper service to interact with the Feder8 platform The configuration provides the context in which the analysis is performed

save_dataframe_as_csv_result(*df: DataFrame*, *filename: str*, *zeppelin_context=None*)

Saves the given data frame as CSV file for the script with the given UUID

save_dataframe_as_json_result(*df: DataFrame*, *filename: str*, *zeppelin_context=None*)

Saves the given data frame as JSON file for the script with the given UUID

save_dictionary_as_json_result(*dd: dict*, *filename: str*, *zeppelin_context=None*)

Saves the given data dictionary as JSON file for the script with the given UUID

2.6 Feder8Utils

2.7 ScriptUuidFinder

2.8 TokenContext

class pyfeder8.TokenContext.**TokenContext**(*access_token*, *id_token*, *refresh_token*, *creation_time*)

Represents a JWT token for Feder8

2.9 TokenContextProvider

class pyfeder8.TokenContextProvider.**TokenContextProvider**(*configuration: Configuration*)

Helper class to retrieve a Feder8 token

get_token_context() → *TokenContext*

Retrieves a TokenContext from any of the available sources :return: TokenContext

CHAPTER

THREE

EXAMPLES

3.1 Retrieve local configuration

```
from pyfeder8 import Feder8Utils

configuration = Feder8Utils.get_feder8_configuration()
```

3.2 Get local database connection details

```
from pyfeder8 import Feder8Utils

configuration = Feder8Utils.get_feder8_configuration()
db_connection_details = Feder8Utils.get_db_connection_details(configuration)
```

3.3 Get local database engine

```
from pyfeder8 import Feder8Utils

configuration = Feder8Utils.get_feder8_configuration()
db_connection_details = Feder8Utils.get_db_connection_details(configuration)
db_engine = Feder8Utils.get_db_engine(db_connection_details, admin=True):
```

3.4 Get Feder8 token by use of a local configuration

```
from pyfeder8 import Feder8Utils

configuration = Feder8Utils.get_feder8_configuration()
token = Feder8Utils.get_feder8_token(configuration)
```

3.5 Get Feder8 token by use of a username and password

```
configuration = ConfigurationBuilder.build_configuration(TherapeuticDomain.HONEUR,
                                                          Environment.PRD, hostname=
↪"localhost")
token_context = TokenContextProvider(configuration)._get_token_context_from_central_
↪token_endpoint(username=username, api_key=api_key)
```

3.6 Save local results

```
def get_feder8_script_uuid():
    return ScriptUuidFinder.find_script_uuid_in_env()

def save_results(source_name, dd, results_dir, configuration):
    script_dir = os.path.dirname(os.path.realpath('__file__'))
    if not os.path.exists(results_dir):
        os.makedirs(results_dir)
    cur_date = str(datetime.datetime.today().strftime('%Y%m%d'))
    filename = f'{source_name}_results_{cur_date}.json'
    filename = os.path.join(script_dir, f'{results_dir}/{filename}')

    script_uuid = get_feder8_script_uuid()
    token_context = get_feder8_token(configuration)

    if script_uuid and token_context:
        logging.info(f"Save {filename}")
        catalogue_client = CatalogueClient(configuration)
        result_uuid = catalogue_client.save_dictionary_as_json_script_result(
            script_uuid, dd, filename=filename, token_context=token_context)
        logging.info(f"Result successfully saved!  UUID: {result_uuid}")
    else:
        logging.info("No script uuid provided or token context could be found, Data␣
↪Profiler results cannot be shared!")
        with open(filename, "w") as json_file:
            json.dump(dd, json_file)
        logging.info(f"{filename} successfully saved locally.")
```

3.7 List study results

```
catalogue_client = CatalogueClient(configuration)
list_of_results = catalogue_client.list_script_results(study_id=study_id,
                                                       script_uuid=script_uuid,
                                                       latest_version_only=True,
                                                       token_context=token_context)
```

3.8 Download study results

```
for index, row in list_of_results.iterrows():
    file_in = catalogue_client.download_file_with_key(row['storageFileKey'], token_
↪context)
    with open(row['filename'], "wb") as json_file:
        json_file.write(file_in)
```

3.9 Distributed Analytics - number of patients per site

```
configuration = ConfigurationBuilder.build_configuration(TherapeuticDomain.HONEUR,␣
↪Environment.UAT, hostname="localhost")
image_repo = configuration.central_service_connection_details.image_repo
token_context = TokenContextProvider(self.configuration)._get_token_context_from_central_
↪token_endpoint(username=username, password=password)
da_client = DistributedAnalyticsClient(self.configuration)
study = "distributed-analytics-test-uat"
request_uuid = str(uuid.uuid4())
organizations = ["TestOrg1", "TestOrg2"]
image_name_tag = image_repo + "/script/run-query-and-export-to-csv:1.0.0"
docker_request = DockerRequest(name="person-count",
                               description="Run person count query",
                               image_name_tag=image_name_tag,
                               env_vars={ "DB_HOST": "postgres", "DB_DATABASE_NAME":
↪"OHDSI", "DB_OMOPCDM_SCHEMA": "omopcdm", "FEDER8_SQL_QUERY": "SELECT count(*) as␣
↪person_count FROM person"},
                               volumes={"shared": "/var/lib/shared"})
responses = da_client.run_docker_image(study=self.study,
                                       organizations=organizations,
                                       docker_request=docker_request,
                                       request_uuid=request_uuid,
                                       token_context=token_context)
print(responses)
for org in organizations:
    if not responses[org].get("errorResponse"):
        df = pd.read_csv(StringIO(responses[org].get("payload")))
        print(org + ":\n" + df.to_string())
```

3.10 Distributed Analytics - distributed average

```
import json
import logging
import uuid

from pyfeder8.TokenContext import TokenContext
from pyfeder8.config.Configuration import Configuration
from pyfeder8.distributed_analytics.DistributedAnalyticsClient import␣
↪DistributedAnalyticsClient
```

(continues on next page)

(continued from previous page)

```
from pyfeder8.distributed_analytics.DockerRequest import DockerRequest


def run_distributed_average(configuration: Configuration,
                            study: str,
                            organizations: [],
                            sql_query: str,
                            column_name: str,
                            token_context: TokenContext = None):
    da_client = DistributedAnalyticsClient(configuration)
    image_repo = configuration.central_service_connection_details.image_repo

    distributed_request_uuid = str(uuid.uuid4())

    query_request_dict = {}
    sql_query_docker_request_name_prefix = "distributed-average-data-preparation-"
    for organization in organizations:
        sql_query_docker_request = DockerRequest(name=sql_query_docker_request_name_
↪prefix + organization,
                                                 description="Prepare data for␣
↪distributed average",
                                                 image_name_tag=image_repo + "/
↪distributed-analytics/run-query-and-export-to-csv:1.0.0",
                                                 env_vars={"DB_HOST": "postgres", "DB_
↪DATABASE_NAME": "OHDSI",
                                                           "DB_OMOPCDM_SCHEMA": "omopcdm
↪",
                                                           "FEDER8_SQL_QUERY": sql_query}
↪,
                                                 volumes={"shared": "/var/lib/shared"})

        request_messages = da_client.create_and_send_request_messages(study,␣
↪[organization],
                                                                      sql_query_docker_
↪request,
                                                                      distributed_
↪request_uuid,
                                                                      token_context)
        query_request_dict[organization] = request_messages.get(organization)

    da_client.wait_for_responses(distributed_request_uuid, organizations, token_context)

    for organization in organizations:
        request_messages = query_request_dict[organization].get("requestMessages")
        sql_query_docker_request_name = sql_query_docker_request_name_prefix +␣
↪organization
        sql_query_docker_request = [m for m in request_messages if sql_query_docker_
↪request_name in m.get("payload")][0]
        r_uuid = sql_query_docker_request.get("uuid")
        database_uri = "/home/feder8/data/" + distributed_request_uuid + "/" + r_uuid +
↪"/result.csv"
        distributed_average_docker_request = DockerRequest(name="distributed-average",
```

(continues on next page)

(continued from previous page)

```
                                                    description="Distributed average",
                                                    image_name_tag=image_repo + "/
↪distributed-analytics/vantage6-distributed-average:1.0.0",
                                                    env_vars={ "COLUMN_NAME": column_name,
↪"DATABASE_URI": database_uri }, volumes={})
        da_client.create_and_send_request_messages(study, [organization],
                                                   distributed_average_docker_request,
                                                   distributed_request_uuid,
                                                   token_context)

    responses = da_client.wait_for_responses(distributed_request_uuid, organizations,␣
↪token_context)

    # Now we can combine the partials to a global average.
    global_sum = 0
    global_count = 0
    for organization in organizations:
        logging.info(organization + ":\n" + responses[organization].get("payload"))
        output = json.loads(responses[organization].get("payload"))
        global_sum += output["sum"]
        global_count += output["count"]

    if global_count <= 0:
        logging.warning("Global count should be larger than 0!")
        return None

    return {"average": global_sum / global_count}
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

T