NeuroLake Hydra Development Kit

Query Developer Documentation

Welcome to NeuroLake! We are happy to bring you to our community of developers! Here at NeuroLake we collect data across all sorts of internet sources and apply our intelligence to build solutions that predict the future! Our platform has three major pillars:

- Collect data
- Analyze and transform data
- Provide the transformed data

Here in this documentation, we will focus on the first pillar: **Collecting Data**! To collect our data we use a set of web scrapers known as *queries*. Basically, a *query* is a piece of code written in Python that accesses a place on the web and retrieves information.

Queries run in one of our platform's solutions called *NeuroLake Hydra*. It is responsible for orchestrating everything needed for a *query* to execute. In *Hydra*, we take care of complex processes such as error retrying, block avoiding strategies, query execution speed control, etc. When you execute a Hydra query you don't need to worry about any of those scary things! We take care of that for you, so you can focus on what matters: **Build your Hydra Query**

Now we will guide you through the process of developing your first query. Let's go!

Building your own Hydra Query

What do you need installed?

First things first! To build a Hydra Query you need to meet the following requirements:

- Python 3.6 installed
- PIP updated to the latest version
- Any Linux distribution

Pretty simple, right? With all the requirements installed, all you need to do is to execute the requirements.txt file that comes shipped with your Hydra Development Kit:

```
$ cd ~/Your_Hydra_Folder/
$ pip install -r requirements.txt
```

Using the Hydra Development Kit

Hydra Development Kit's mission is to make building Hydra Queries as easy as possible! Using it is pretty simple! Let's have a look at the package's structure.

```
├── chrome.deb
├── chromedriver
├── query.py
├── Readme.md
├── requirements.txt
└── utils
    ├── Lake_Enum.py
    ├── Lake_Exceptions.py
    ├── Lake_Utils.py
```

The only file you need to worry about is `query.py`. We will talk about it later, but for now, let's review our package structure by order.

The "Utils" package

The Utils package contains all the methods needed by a *hydra query* to work. Using the utils methods ensures that your code is following our development standards. The following methods are provided by the utils package:

Method	Description
save_data	Saves the provided content in the Hydra Structure. This method ensures that the data you are saving is stored according to NeuroLake patterns
generate_file_name	generates a file name for the file that will be saved by `save_data`
generate_timestamp	Creates a new timestamp string in the pattern used by NeuroLake applications

Using Lake_Exceptions

The last file in our package is `Lake_Exceptions.py`. Since no code is immune to failure, we need to make sure your *hydra query* speaks the same language as our architecture when it comes to exception raising.

The following exceptions are available as part of our `Lake_Exceptions`package:

Exception	Use when
BlockException	Use it when your query has encountered any situation where it is impossible to move forward. For example Captcha, the website is not responding, the machine IP was black mailed
HttpTimeoutException	Your requisition to the target host is taking too long to arrive
PageNotLoadedException	The page is not loading
UnexpectedResultException	The page loaded differently from expected
CriticalErrorException	**ANY** exception that is not one of the above

The query template

The query template is the actual file you are going to be developing. It a pre-filled python code that basically does nothing by itself, but with your help, will become a brand new *Hydra Query*!

The template itself contains documentation that will help you use it. But basically, you need to fill the `query_execution` with your code.

The HydraMetadata section Your query file contains all the information needed by the architecture to execute it. A very important section is the *HydraMetadata* section. This JSON object is translated by the architecture during the API query register and tells it how your query works and what it will need to execute properly. Here is an explanation about the content of the HydraMetadata section:

Key	Type	Description
version	string	The version of your query. We follow a simple version of the Semantic Versioning pattern. This means that your query version must follow the pattern MAJOR.MINOR.PATCH
requirements	list [string]	a list of python libraries required by your query to execute. This list will be passed to the `pip install -r` command.
developer_contact	string	A valid email. It is important to provide a valid email because this is how we will contact you if anything go wrong with your query registration
host	string	This is the starting point of your query. This is the domain in the format https://www.example.com
timeout	integer	This is the maximum time you query takes to execute completely
selenium_usage	string	if you set this to "true", the hydra architecture will build a selenium webdriver specially tunned to execute with the best performance in the AWS EC2 environment
query_name	string	This is the Codename for your query. It is composed by a Three letters and Three numbers long. For example: TST001. We will provide you with the rules for naming your query

Your `query_execution` method receives two arguments as it's inputs:

Argument	Description
input_data	A dictionary containing the input data to be executed by your query. When you provide the Hydra engine with a file containing multiple lines of input, each hydra worker receives one line as a dictionary `{"query_key":"input_content"}`
properties	This dictionary contains the properties needed by your query to work properly in our architecture. Currently, the avaliable properties are `timeout` (number in seconds that you query should take to execute) and `driver` (if you provide the flag "selenium_usage" in your metadata, you will receive a properly configured webdriver to use in your execution)

The result of your query

Your `query_execution` method must be responsible for collecting and processing the data from the web source. You must be responsible for extracting the information and structuring it in a JSON object (python dict). We call this process *parsing*. Once you finished parsing your data, you must return it as a dict.

For more information, refer to the example `query.py` file that comes shipped with this development kit.

##Testing your Hydra Query Every good code should be tested! In Hydra we only accept queries with at least one test implemented. Your `query.py` file comes with a prefilled `test_request` method. This method basically calls your query main function and verifies if the result (returned bu your `query_execution` method) is a python dict. You should improve this method and build others in order for it to test your query's business logic.

To test your hydra query, simply go to your hydra dev kit home folder and type:

```
pytest query.py
```

this will execute the pytest framework and start the tests for your hyra query file

---

##Deploying your Hydra Query

Once your code is developed and testes, you must deploy it to our infrastructure. Here at Hydra, we provide you with a REST API that contains methods for you deploy and start your query executions.

###Query Registration To register your query, use the `/register_hydra_query` method. You will provide the python file developed by you and the Hydra API will perform all the validations to make sure your code follows our development standards. If everything went OK, you will receive a success email.

###Describe your query registration Once you have finished your query registration, you can use the `/describe_query_registry` method to check if you query version was registered and it is ready for use.

###Start your execution If you have a working query version registered, you can start a new execution by using the `/start_execution` method. You must provide an AWS s3 path with all the data you aim to execute you query on. Once your execution is finished, you should have the processed data avaliable for you to use with the Flame framework.

##Hydra API Documentation You will find the documentation for the hydra API in the swagger definition shipped with this Development Kit