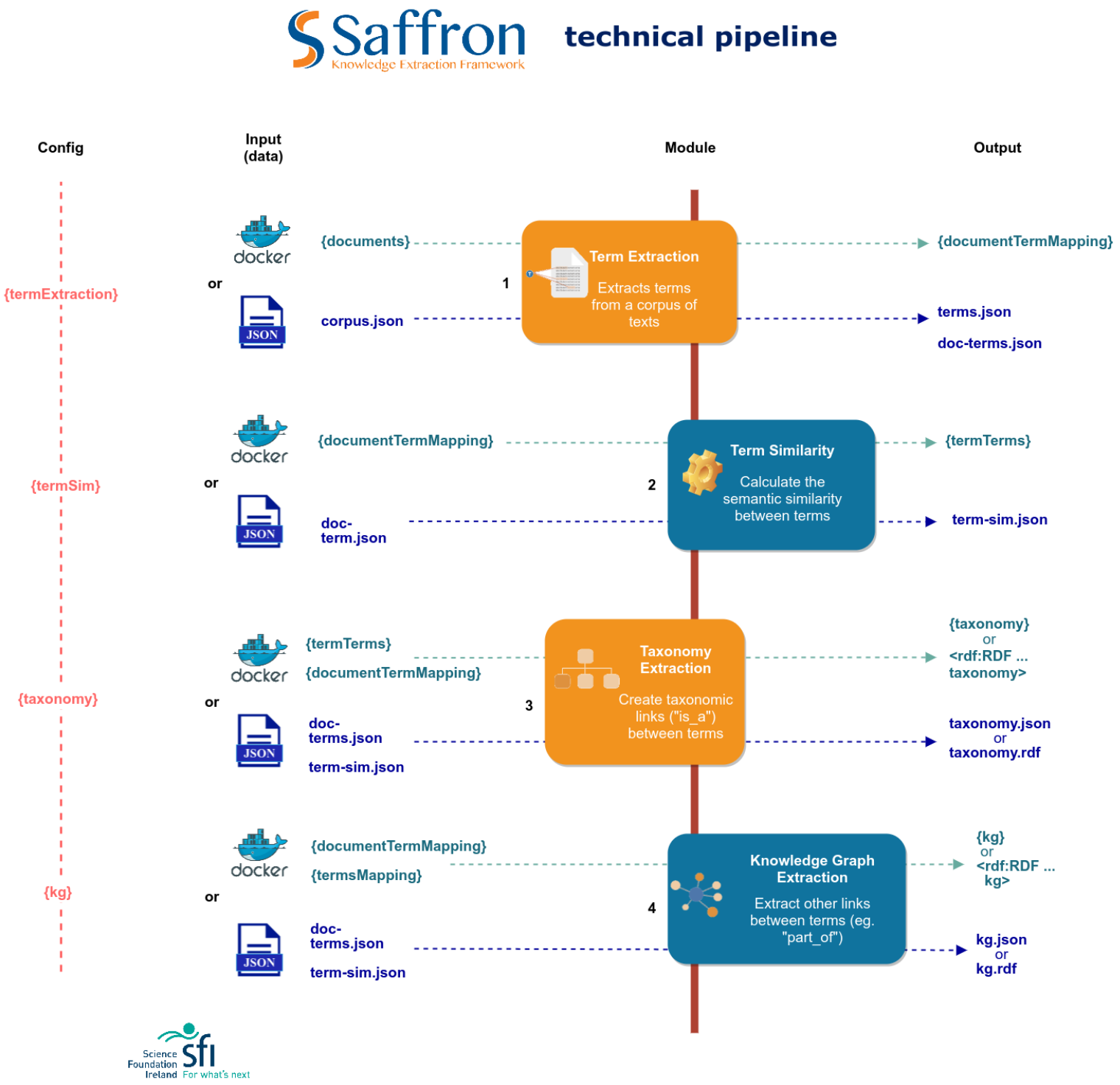
Saffron Documentation

-

Using Docker

1. Technical diagram

https://app.diagrams.net/#G1Ha-Up0j2s34eLBNdlbLh74026IUNHcgx



2. Install and deploy Docker

In order to use Docker in Saffron, **Docker** and **Docker-compose** need to be installed. Install them according to your operating system using the installation guidelines from the links below:

- Docker: https://docs.docker.com/get-docker/ (for Ubuntu users, we recommend installing using the repository)
- Docker-compose: https://docs.docker.com/compose/install/

After installing, Docker needs to be deployed and running. Here are the instructions:

1. Go to your main Saffron folder (considering Saffron is installed)
   ```
   cd saffron
   ```

2. Build all Docker components:
   ```
   docker-compose build
   ```

3. Run docker-compose (let it run in the background):
   ```
   docker-compose up -d
   ```
   (`-d` at the end of the command should detach and run in the background)

4. Now all components should be built and stored in the Docker repository locally. You can check what is running by using the following command:
   ```
   docker ps
   ```

3. Modules running configurations

Input config	Input data	Module	Output
"termExtraction" : { "threshold" : 0.0, "maxTopics" : 100, "ngramMin" : 1, "ngramMax" : 4, "minTermFreq" : 2, "maxDocs" : 2147483647, "method" : "voting", "features" : ["comboBasic", "weirdness", "totalTfIdf", "cValue", "residualIdf"], "corpus" : "${saffron.home}/models/wiki-terms.json.gz", "baseFeature" : "comboBasic", "posModel" : "${saffron.home}/models/en-pos-maxent.bin", "tokenizerModel" : null, "lemmatizerModel" : "${saffron.home}/models/en-lemmatizer.dict.txt", "stopWords" : null, "preceedingTokens" : ["NN", "JJ", "NNP", "NNS"], "middleTokens" : ["IN"], "headTokens" : ["NN", "CD", "NNS"], "headTokenFinal" : true, "blacklist" : [ ], "blacklistFile" : null, "oneTopicPerDoc" : false },	corpus.json	term-extraction	terms.json doc-terms.json
termSim" : { "threshold" : 0.1, "topN" : 50	doc-terms.json	term-similarity	term-sim.json
"authorTerm": { "topN": 1000,	corpus.json doc-terms.json	author-extraction	author-terms.json

"minDocs": 1			
	author-terms.json	author-consolidation?	author-terms.json
"authorTerm": { "topN": 1000, "minDocs": 1	author-terms.json	author-connection	author-terms.json
"authorSim" : { "threshold" : 0.1, "topN" : 50	author-terms.json	author-similarity	author-sim.json

"taxonomy" : {
"negSampling" : 5.0,
"features" : null,
"modelFile" :
"${saffron.home}/models/default.json",
"search" : {
"algorithm" : "greedy",
"beamSize" : 20,
"score" : "simple",
"baseScore" : "simple",
"aveChildren" : 3.0,
"alpha" : 0.01
}
},
 	
	doc-terms.json
	term-sim.json
		taxonomy-extraction	Taxonomy.json
			(if config
			"returnRDF": false)
			or
			Taxonomy.rdf
			(if config
			"returnRDF": true)

"kg" : { "kerasModelFile" : "${saffron.home}/models/model_keras.h5", "bertModelFile" : "${saffron.home}/models/bert_model/", "synonymyThreshold" : 0.5, "meronomyThreshold" : 0.25, "enableSynonymyNormalisation": true	doc-terms.json term-sim.json	kg-extraction	Kg.json (if config "returnRDF": false) or Kg.rdf (if config "returnRDF": true)

4. Run individual Saffron modules using Docker

4.1. Send a request

A/ Modules URLs

To run each individual module, a REST request has to be sent. It is possible to use a REST enabled application, like Postman for example, or send the requests as CURL requests. Here is a list of the different URLs available to send requests to, corresponding to the different Saffron modules (described in more details below):

- http://0.0.0.0:9001/api/v1/term-extraction
- http://0.0.0.0:9002/api/v1/author-consolidation
- http://0.0.0.0:9002/api/v1/author-connection

- http://0.0.0.0:9002/api/v1/author-similarity
- http://0.0.0.0:9004/api/v1/term-similarity
- http://0.0.0.0:9002/api/v1/taxonomy-extraction
- http://0.0.0.0:9002/api/v1/kg-extraction

It is possible to chain the queries together manually to create a pipeline (and link the output into input required for each module, see below). Note that some modules are dependent on others, which need to be run prior to it (they use the output of other modules as input).

B/ Docker request structure

Generally the structure of a request consists of the following main JSON nodes:

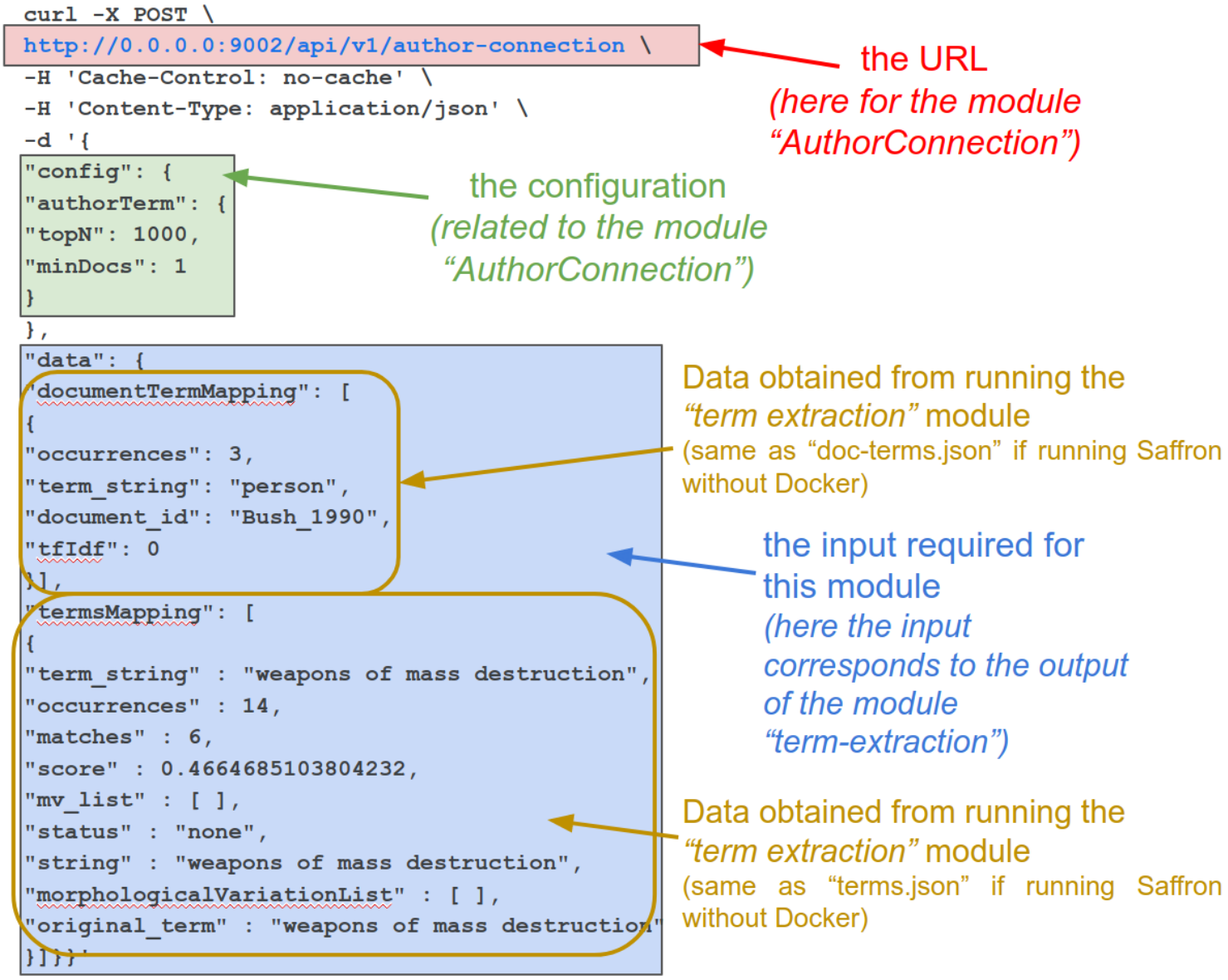
```
{
"config": {}, "data": {}
}
```

- The "**config**" node contains the specific config for the operation (module) you are carrying out. See below for more details for the configuration of each of them.
- The "**data**" node should contain what is needed as **input** for the module to perform. See below for more details on what is required for each module.

C/ Send the request

The request can be sent using a CURL as a POST request. The request needs:
- the URL to the module that needs to be run
- 'Content-Type: application/json'
- the configuration node, and the data node (as described above) (a) either passed using the **-d** argument **within the request itself**, or (b) as a **separate JSON file**.

(a) Below is an example of request the can be sent with the configuration and the data inline using CURL (example with the Author Connection module):



(b) Below is an example of request done by **passing the configuration and the data as a separate .JSON file** (example for the kg extraction module, with this request JSON file):

(c)

```
curl --location --request POST --header 'Content-Type: application/json' --data @/your_path_to_the_request.json 'http://0.0.0.0:9002/api/v1/kg-extraction'
```

(Note the @ before `your_path_to_the_request.json` is required)

Below are more details on each module that can be individually called using Docker.

4.2. Individual modules

For each individual module, we describe here what should be given in the “config” node and the “data” node of the Docker request. In the section “output”, we describe the response obtained after running the request. The format of the response returned is as JSON format.

4.2.1. Term-extraction

URL: http://0.0.0.0:9001/api/v1/term-extraction

4.2.1.1. “config” node

The term extraction module configuration is needed here. It is described here (see an example below)

```
{
  "termExtraction" : {
    "threshold" : 0.0,
    "maxTopics" : 100,
    "ngramMin" : 1,
    "ngramMax" : 4,
    "minTermFreq" : 2,
    "maxDocs" : 2147483647,
    "method" : "voting",
    "features" : [ "comboBasic", "weirdness", "totalTfIdf", "cValue", "residualIdf" ],
    "corpus" : "${saffron.home}/models/wiki-terms.json.gz",
    "baseFeature" : "comboBasic",
    "posModel" : "${saffron.home}/models/en-pos-maxent.bin",
    "tokenizerModel" : null,
    "lemmatizerModel" : "${saffron.home}/models/en-lemmatizer.dict.txt",
    "stopWords" : null,
    "preceedingTokens" : [ "NN", "JJ", "NNP", "NNS" ],
    "middleTokens" : [ "IN" ],
    "headTokens" : [ "NN", "CD", "NNS" ],
    "headTokenFinal" : true,
    "blacklist" : [ ],
    "blacklistFile" : null,
    "oneTopicPerDoc" : false
  },
```

4.2.1.2. “data” node

The corpus to run the module on, in the **Saffron corpus json format** described here (example below). The main node of the JSON is “documents”.

```
{
  "documents" : [ {
    "file" : "${saffron.home}/examples/presidential_speech_dataset/presidential_speech_corpus_texts/Bush_1989.txt",
    "id" : "Bush_1989.txt",
    "name" : "George H.W. Bush's 1st State of the Union Address",
    "mime_type" : "text/plain",
    "authors" : [ "George H.W. Bush" ],
    "metadata": { "year": "1989" },
    "date": "1989"
  }, {
    "file" : "${saffron.home}/examples/presidential_speech_dataset/presidential_speech_corpus_texts/Bush_1990.txt",
    "id" : "Bush_1990.txt",
    "name" : "George H.W. Bush's 2nd State of the Union Address",
```

4.2.1.3. Output

The response of the request will be a JSON-formatted output, containing **two main nodes**: "**documentTermMapping**" and "**termsMapping**", as shown in the example below:


```
{
    "documentTermMapping": [
        {
            "occurrences": 2,
            "tfidf": 8.761109928124851,
            "document_id": "2655",
            "term_string": "fingerprint"
        },
        {
            "occurrences": 1,
            "tfidf": 4.092872891610645,
            "document_id": "3414",
            "term_string": "time"
        },
    ],
    "termsMapping": [
        {
            "occurrences": 71,
            "score": 0.022156686234940968,
            "mv_list": [],
            "string": "time",
            "original_term": "time",
            "term_string": "time",
            "matches": 68,
            "status": "none"
        },
        {
            "occurrences": 61,
            "score": 0.09683753952954093,
            "mv_list": [],
            "string": "fingerprint",
            "original_term": "fingerprint",
            "term_string": "fingerprint",
            "matches": 51,
            "status": "none"
        },
    ]
}
```


4.2.1.4. Example

See joined file *term_extraction_rq.json* for an example of request file for this module

4.2.2. Term-similarity

URL: http://0.0.0.0:9004/api/v1/term-similarity
Note: The *term extraction* module needs to be run before the *term similarity* module.

4.2.2.1. "config" node

The *term similarity* module configuration (content of the "termSim" element) is needed here. It is described here (see an example below)

```
"config": {
    "termSim": {
        "threshold": 0.1,
        "topN": 50
    }
},
```

4.2.2.2. "data" node

The output from the *term-extraction* module is needed here (ie. the content of doc-terms.json, using the "documentTermMapping" node)

```
"data": {
    "documentTermMapping": [
        {
            "occurrences": 2,
            "tfidf": 8.761109928124851,
            "document_id": "2655",
            "term_string": "fingerprint"
        },
        {
            "occurrences": 1,
            "tfidf": 4.092872891610645,
            "document_id": "3415",
            "term_string": "time"
        },
    ]
}
```

4.2.2.3. Output

The response of the request will be a JSON-formatted output, containing **one main node**: **"termTerms"**, as shown in the example below:

```
"termTerms": [

    {
        "term1_id": "password",
        "term2_id": "phone number",
        "similarity": 0.17161692805605763,
        "status": null
    },
    {
        "term1_id": "password",
        "term2_id": "mobile number",
        "similarity": 0.18058421702886368,
        "status": null
    },
    {
        "term1_id": "fingerprint",
        "term2_id": "screen",
        "similarity": 0.11734292976977152,
        "status": null
    },
    {
        "term1_id": "fingerprint",
        "term2_id": "payment",
        "similarity": 0.20460160615732953,
        "status": null
    },
```


4.2.2.4. Example

See joined file *term_similarity_rq.json* for an example of request file for this module

4.2.3. Taxonomy-extraction

URL: http://0.0.0.0:9005/api/v1/taxonomy-extraction
Note: The *term extraction* and the *term similarity* modules need to be run before the *taxonomy* module.
The output will either be as JSON format (if config "returnRDF": false) or XML/RDF (if config "returnRDF": true)

4.2.3.1. “config” node

The taxonomy module configuration (content of the “taxonomy” element) is needed here. It is described here (see an example below)

```
"taxonomy" : {
  "returnRDF": true,
  "negSampling" : 5.0,
  "features" : null,
  "modelFile" : "${saffron.home}/models/default.json",
  "search" : {
    "algorithm" : "greedy",
    "beamSize" : 20,
    "score" : "simple",
    "baseScore" : "simple",
    "aveChildren" : 3.0,
    "alpha" : 0.01
  }
```

4.2.3.2. “data” node

The output from the *term-extraction* module is needed here (ie. the content of doc-terms.json, using the “documentTermMapping” node)

- The output from the *term-extraction* module (e. the content of docTerms.json, using the “documentTermMapping” node)
- The output from the *term-similarity* module (ie. the content of the term-sim.json, using the “termTerms” node).

```
"data": {
    "documentTermMapping": [
        {
            "occurrences": 1,
            "tfidf": 4.02192115563836,
            "document_id": "652",
            "term_string": "password"
        },
        {
            "occurrences": 1,
            "tfidf": 1.7152348949000995,
            "document_id": "1681",
            "term_string": "phone number"
        },

        {
            "occurrences": 1,
            "tfidf": 2.691979731069601,
            "document_id": "3499",
            "term_string": "mobile number"
        },

    ],
    "termTerms": [

        {
            "term1_id": "password",
            "term2_id": "phone number",
            "similarity": 0.17161692805605763,
            "status": null
        },
        {
            "term1_id": "password",
            "term2_id": "mobile number",
            "similarity": 0.18058421702886368,
            "status": null
        }
    ]
}
```


4.2.3.3. Output

The response of the request will depend on the setting of the configuration field: "**returnRDF**".

- If set to *true*: The output will be be a RDF-XML formatted output, as shown in the example below:
- If set to *false*: The output will be be a JSON-formatted output, containing **one main node**: **"taxonomy"**, as shown in the example below:

4.2.3.4. Example

See joined file *taxonomy_extraction_rq.json* for an example of request file for this module

4.2.4. Kg-extraction

URL: http://0.0.0.0:9005/api/v1/kg-extraction
Note: The *term extraction* and the *term similarity* modules need to be run before the *kg* module.
The output will either be as JSON format (if config "returnRDF": false) or XML/RDF (if config "returnRDF": true)
Note that the process can take several hours on big files.

4.2.4.1. "config" node

The knowledge graph module configuration (content of the "kg" element) is needed here. It is described here (see an example below)

```
"kg" : {
  "kerasModelFile" : "${saffron.home}/models/model_keras.h5",
  "bertModelFile" : "${saffron.home}/models/bert_model/",
  "synonymyThreshold" : 0.5,
  "meronomyThreshold" : 0.25,
  "enableSynonymyNormalisation": true
}
```

4.2.4.2. "data" node

The output from the *term-extraction* module is needed here (ie. the content of doc-terms.json, using the "documentTermMapping" node)

- The output from the *term-extraction* module (e. the content of docTerms.json, using the "documentTermMapping" node)
- The output from the *term-similarity* module (ie. the content of the term-sim.json, using the "termTerms" node).

```
"data": {
    "documentTermMapping": [
        {
            "occurrences": 1,
            "tfidf": 4.02192115563836,
            "document_id": "652",
            "term_string": "password"
        },
        {
            "occurrences": 1,
            "tfidf": 1.7152348949000995,
            "document_id": "1681",
            "term_string": "phone number"
        },

        {
            "occurrences": 1,
            "tfidf": 2.691979731069601,
            "document_id": "3499",
            "term_string": "mobile number"
        },

    ],
    "termTerms": [

        {
            "term1_id": "password",
            "term2_id": "phone number",
            "similarity": 0.17161692805605763,
            "status": null
        },
        {
            "term1_id": "password",
            "term2_id": "mobile number",
            "similarity": 0.18058421702886368,
            "status": null
        }
    ]
}
```


4.2.4.3. Output

The response of the request will depend on the setting of the configuration field: "**returnRDF**".

5. If set to *true*: The output will be be a RDF-XML formatted output, as shown in the example below:
6. If set to *false*: The output will be be a JSON-formatted output, as shown in the abstract example below:

```
{
  "taxonomy": {
    "root": "HEAD_TERM",
    "score": 0,
    "linkScore": 0,
    "children": [
      {
        "root": "cold war",
        "score": 0.028017059553830083,
        "linkScore": "NaN",
        "children": [
          {
            "root": "trillion",
            "score": 0.00091194706630528,
            "linkScore": 0.6458660960197449,
            "children": [
              {
                "root": "company",
                "score": 0.1305505208949439,
                "linkScore": 1,
                "children": [],
                "status": "none",
                "parent": null
              },
  "synonymyClusters": [
    [
      "rest",
      "american person"
    ]
  ],
  "ontonomy": {
    "relations": []
  },
  "synonymyRelations": [
    {
      "source": "american person",
      "target": "rest",
      "type": "synonymy"
    }
  ]
}
```


6.1.1.1. Example

See joined file *kg_extraction_rq.json* for an example of request file for this module

7. Create a Saffron module using Docker

To dockerize a new module:

Pre-requisites:

1. Install Docker: https://docs.docker.com/get-docker/ (for Ubuntu users, we recommend installing using the repository)
2. Install Docker-compose: https://docs.docker.com/compose/install/

Steps to follow:

1. Go to the module home directory and create a file named Dockerfile (case-sensitive. Make sure to name the file exactly and without any extension)
2. For a Python application follow https://runnable.com/docker/python/dockerize-your-python-application for initial setup.
3. For a Java follow https://runnable.com/docker/java/dockerize-your-java-application steps for steps

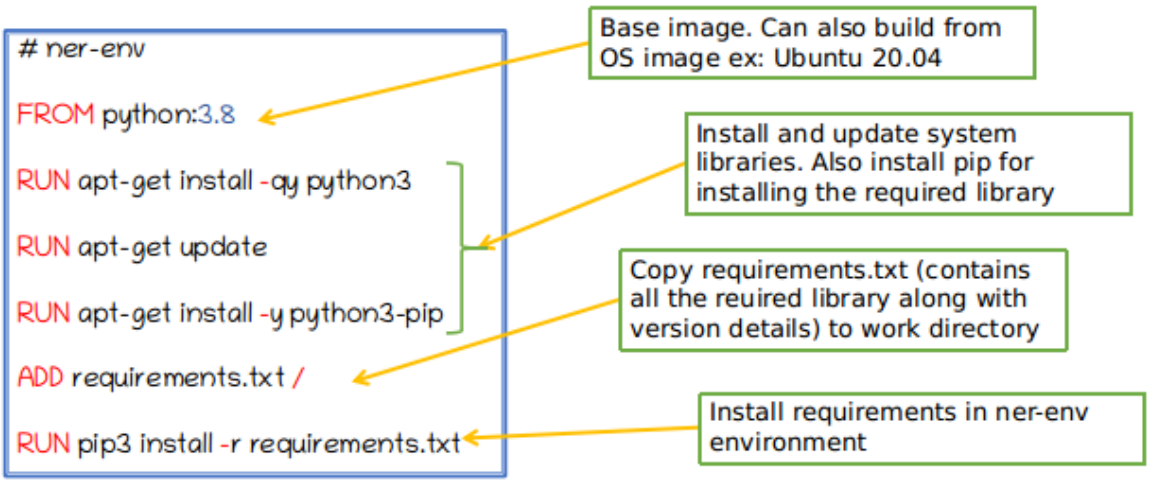
Once you have the Dockerfile ready:

1. Build you application into the docker container
   docker build -t python-app .
2. Run your application from docker container
   docker run python-app

Example Dockerfile:

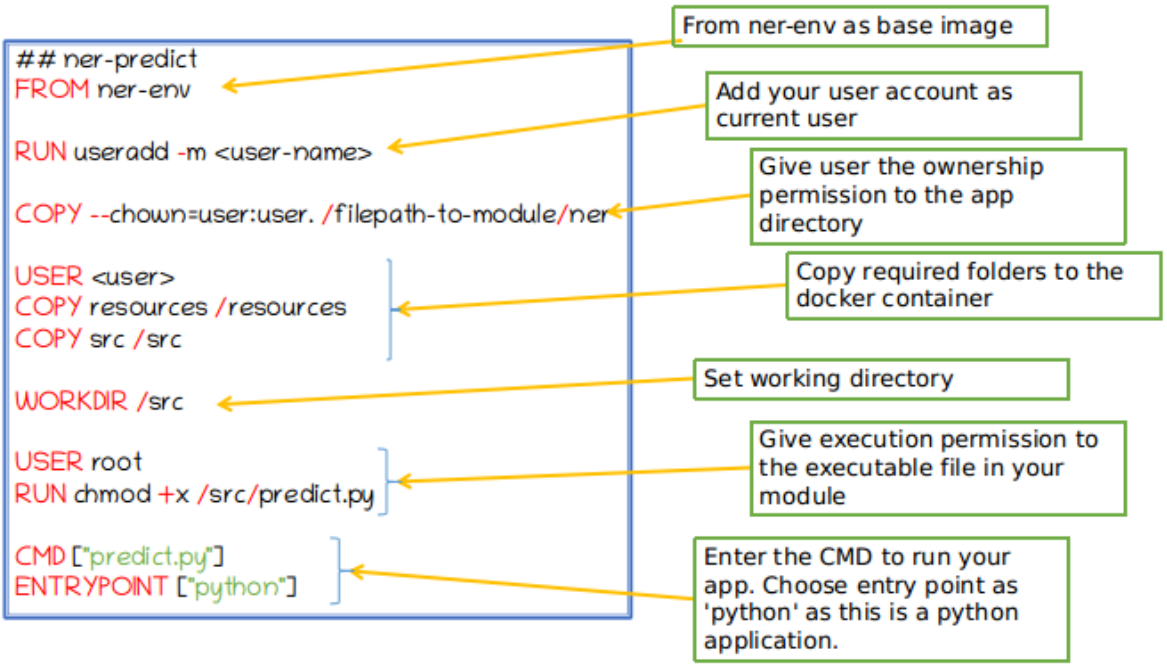
- To avoid installing required libraries multiple times, create an environment in docker and from that environment build your module.
- Unless the requirement changes, there is no need to build the environment each time you push new changes. This method also reduces the app disk space in the docker container.

Environment:



Build: docker build -t ner-env .

App container:



Build: docker build -t ner-predict .
Run: docker run ner-predict