



D3.5 Workflows for NLP services

Author(s): Bernardo Stearns, John McCrae (NUIG)

Date: 30.06.2021



H2020-ICT-29b

Grant Agreement No. 825182

Prêt-à-LLOD - Ready-to-use Multilingual Linked Language Data for
Knowledge Services across Sectors

D3.5 Workflows for NLP services

Deliverable Number: D3.5

Dissemination Level: Public

Delivery Date: 2021-06-30

Version: 1.0

Author(s): Bernardo Stearns, John McCrae

Document History

Version Date	Changes	Authors
2020-06-21	Initial document	Bernardo Stearns
2020-07-09	Final contributions	Bernardo Stearns



Table of Contents

Introduction	4
Teanga	4
Overview	4
Technologies	5
Architecture	8
Workflows	10
Usage	12
References	



Introduction

This document is a report related to the software deliverable D3.5 “Workflows for NLP services”. This report was planned to serve as a guideline of the usage of the workflow software (Teanga) and systems developed in Prêt-à-LLOD as part of the “Workflows for NLP services” component: what are its basic functionalities, what technologies were used, and how they were implemented as an architecture. This is also a starting point to find external links such as github repositories, docker hub, examples and other documentation.

D3.5, Prêt-à-LLOD Workflows addresses the challenge to create “*Workflows for Portable and Scalable Semantic Language Services*”. A protocol, based on semantic markup, is developed to enable language services to be easily connected into multi-server workflows.

The current code base for D3.5 resides in GitHub and its released software resides in Docker Hub. Those repositories are being continuously updated and the latest release of the software can be accessed at any time and issues can be created through GitHub.

- [GitHub \(codebase\)](#)
- [Teanga documentation](#)
- [Teanga in Docker Hub](#)

1. Teanga

With the fast-paced growth of NLP research, a need for efficient and scalable integration and customization of NLP services has emerged as a necessity for sustainable NLP pipelines. Teanga is a workflow manager that helps with that. Users create, explore and execute complex workflows, such as linking data, sentiment analysis and other natural language processing applications by leveraging the power of the Semantic Web to connect services and solid technologies and standards to enable service providers to integrate their service in Teanga. Teanga is distributed as a command line tool that manages the whole architecture and gives users the flexibility to easily install, test and run its software both locally or in servers. Within the command line tool the user can trigger a local user interface where they can create new workflows or execute existing ones visually, assisting less experienced and less technical users.

Teanga is implemented relying on well established standards and technologies such as Docker, OpenAPI specification, Apache Airflow and REST APIs and all concepts in it are a combination of those technologies. Apart from end users which use the command line tool for building workflows, the mentioned technologies enables service providers to reliably integrate and test their service in Teanga.

In the following sections we will highlight the main aspects of the technologies and architectures used in Teanga followed by usage examples.



2. Technologies

Teanga involves a combination of a command line tool, a webserver, a frontend and a scheduler/executor. In this section we will discuss each technology separately for later discussing how those technologies were combined into an integrated architecture.

Airflow

Airflow is an easy to use open source platform created by the Python community to programmatically author, schedule and monitor workflows. Implemented in the Python programming language and it gained popularity (21.9k stars on github) due to being scalable, dynamic, extensible and elegant.

Airflow has a modular architecture and uses a message queue to orchestrate an arbitrary number of workers, enabling it to scale easily. Its pipelines are defined in Python, allowing for writing code that generates pipelines dynamically and easily defines your own operators and libraries. Tied to the work scalability, Airflow also provides an useful UI to monitor, schedule and manage your workflows via a robust and modern web application having always full insight into the status and logs of completed and ongoing tasks.

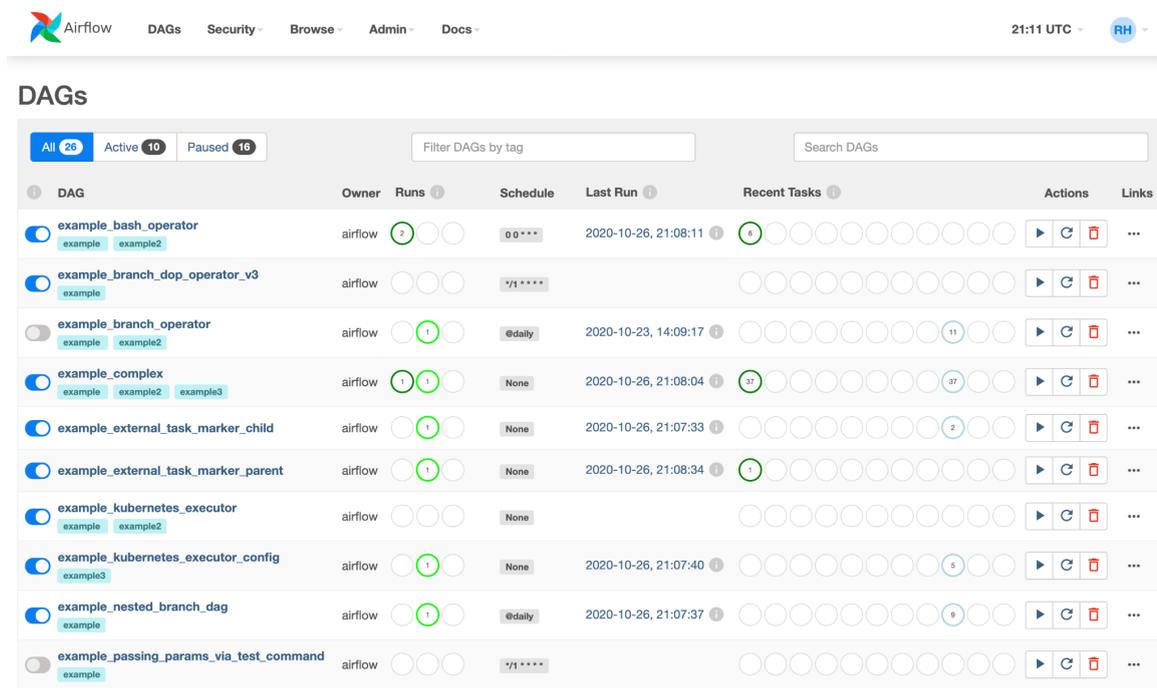


figure 2.1: Airflow list of workflows

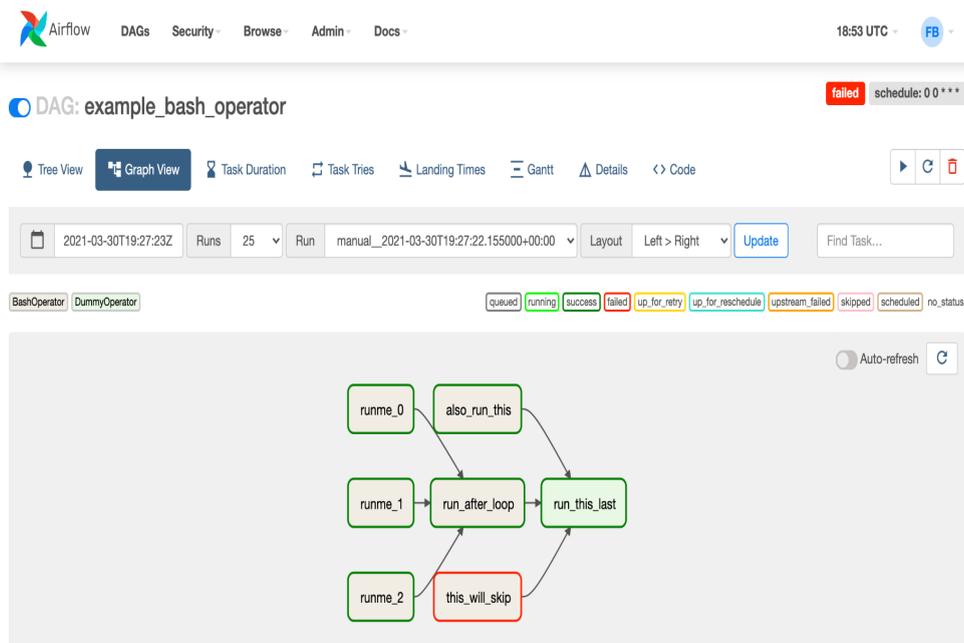


Figure 2.2: Airflow visualization of workflow execution

Airflow has three main concepts: operators, tasks and workflows. A *workflow* is represented as a [DAG](#) (a Directed Acyclic Graph), and contains individual pieces of work called [tasks](#), arranged with dependencies and data flows taken into account. *Operators* are a template for a predefined task, that you can just define declaratively inside your Workflow and connect to its dependencies.

Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

REST APIs

An Application Programming Interface (API) defines the allowed interactions between two pieces of software, just like a user interface defines the ways in which a user can interact with a program.

An API is composed of the list of possible methods to call (requests to make), their parameters, return values and any data format they require (among other things). This is equivalent to how a user's interactions with a mobile phone app are limited to the buttons, sliders and text boxes in the app's user interface.

A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.

In order for an API to be considered RESTful, it has to conform to these criteria:

- A client-server architecture made up of clients, servers, and resources, with requests managed through HTTP.
- Stateless client-server communication, meaning no client information is stored between get requests and each request is separate and unconnected.
- Cacheable data that streamlines client-server interactions.
- A uniform interface between components so that information is transferred in a standard form.
- A layered system that organizes each type of server (those responsible for security, load-balancing, etc.) involves the retrieval of requested information into hierarchies, invisible to the client.

OpenAPI specification

The OpenAPI Specification (OAS) defines a standard, programming language-agnostic interface description for HTTP APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic. When properly defined via OpenAPI, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. Similar to what interface descriptions have done for lower-level programming, the OpenAPI Specification removes guesswork in calling a service.

An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases.

3. Architecture

The architecture for a workflow manager involves many different modules, each responsible for possible action on a workflow. We overall have three layers :

1. **CLI** : responsible for initialization and configuration of the whole architecture
2. **UI/Web Server** : responsible for letting the user do any action visually



3. Backend: responsible for dealing with all the workload generated from user's actions

Those three layers communicate with each other and are set up in a local folder on the user's machine. Everything that is generated in Teanga is generated under this central local folder, enabling technical users to access configuration files, output files and examples.

The UI and the backend are tied together and run on a docker container that is triggered by the CLI locally, in this way we can guarantee that both will work properly in many different environments.

We can see below a diagram of the overall architecture:

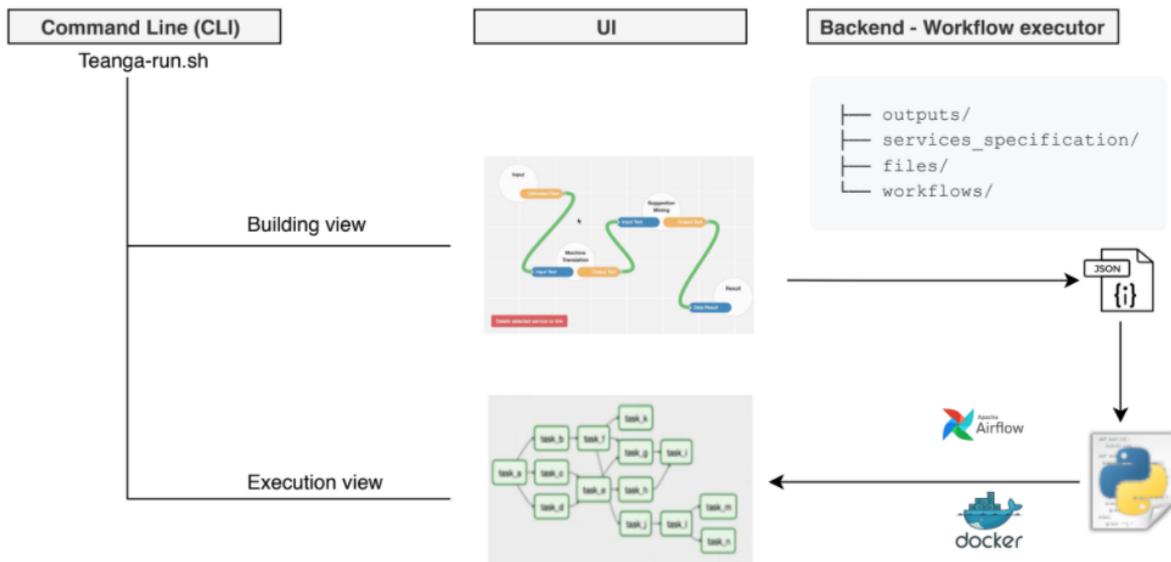


figure 3.1

Having the layers of the architecture in mind, We detail every component's implementation:

Command Line Tool

Overall the command line tool was created to enable user setting up the whole Teanga infrastructure with very few commands. Currently there are two actions, "init" Teanga which sets up all the other components or "run" which executes a given workflow json file and produces outputs. The CLI is completely implemented in bash script (.sh) as it is a very versatile scripting language and due to its simplicity it can be installed in many different environments and operating systems.

Web Server

The web server is the official web server distributed by Apache Airflow extended with a custom Teanga view that lets users create workflows through drag and drop. Once the user creates a workflow it can be accessed through the Airflow workflows list and then it can be executed generating outputs as shown in [Figure 2.1](#) (Airflow section).

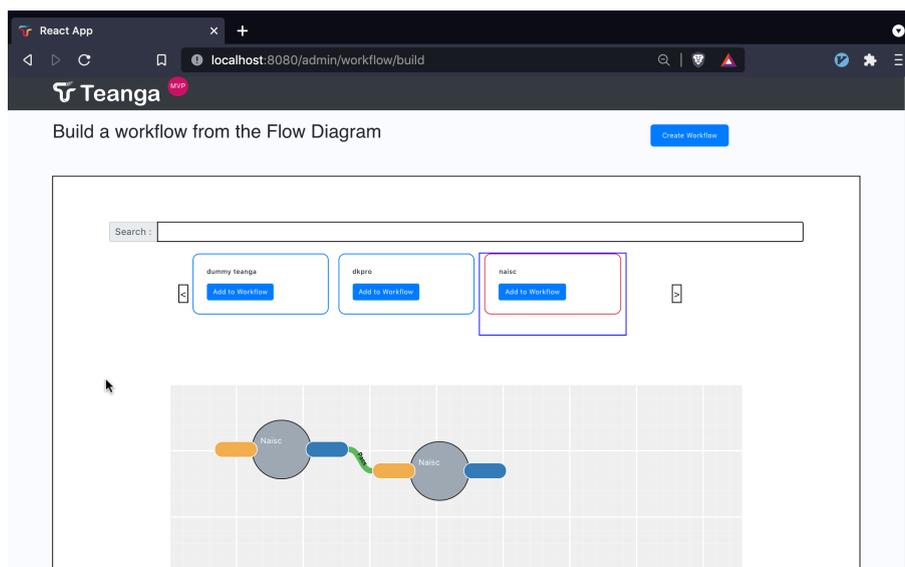


figure 3.2 Teanga UI for creating workflows

Scheduler

Teanga uses the Airflow scheduler as well. It monitors all tasks and DAGs, then triggers the task instances once their dependencies are complete. Behind the scenes, the scheduler spins up a subprocess, which monitors and stays in sync with all DAGs in the specified DAG directory. Once per minute, by default, the scheduler collects DAG parsing results and checks whether any active tasks can be triggered. With this scheduling system whenever a new Workflow (DAG) is created through the Teanga view it will automatically become available in airflow.

Executor

Executors are the mechanism by which task instances get run. Currently the default behavior is set up to run as local and sequence execution, meaning that it will run one task at a time. In the default Airflow installation, this runs everything inside the scheduler, but most production-suitable executors actually push task execution out to workers and could be extended to distribute tasks between servers, potentially improving runtime performance.

Folder of DAG files

A local folder specified by the user where Teanga will create and keep track of existing workflows. It is specified as a configuration in the CLI.

Metadata database

A Postgres database used by the scheduler, executor and web server to store all states of Teanga.

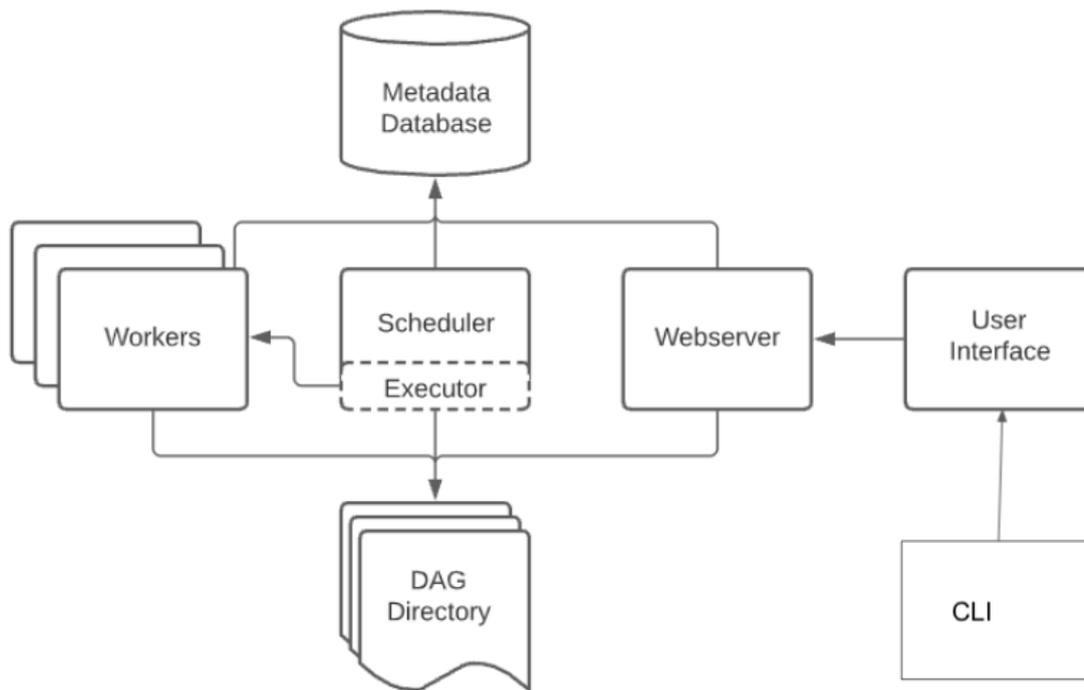


figure 3.3

4. Workflows

Workflows are the main component in Teanga. Using a similar concept of [Airflow DAGS](#) where a sequence of operations(snippets of code) are represented as a graph. In Teanga each operation is a request to a local or remote service which is a Dockerized REST API following the OpenAPI specification. If the service follows the OpenAPI specification, every endpoint will have an operation ID so that by knowing the Docker image information (repository name, image id and image tag) Teanga can access and manage this image as necessary. Once you have that information for every endpoint you want to use in your workflow, you just need to specify the inputs for that endpoint and their dependencies. By default if an input is missing for a step in the workflow, Teanga will check its dependencies inputs and outputs to try to have all the inputs necessary. If this matching fails and inputs still missing the operation will be flagged as failed.

Currently the only way to create a workflow is using the user interface in the view that can be accessed at localhost:8080/admin/workflow/build

Workflows are generated by the UI as JSON files such as in the example below:

```
{
  "1":{
    "operation_id":"upload",
    "input": {
      "id": "left_id",
      "files": ["left.rdf"]
    },
    "repo":"berstearns",
```

```

    "image_id": "naisc",
    "image_tag": "062020",
    "host_port": 8001,
    "container_port": 8080,
    "dependencies": []
  },
  "2": {
    "operation_id": "upload",
    "input": {
      "id": "right_id",
      "files": ["right.rdf"]
    },
    "repo": "berstearns",
    "image_id": "naisc",
    "image_tag": "062020",
    "host_port": 8002,
    "container_port": 8080,
    "dependencies": []
  }
}

```

Teanga make use of RDF technologies and matching strategies to verify if the outputs of services are compatible with the required inputs of the next services in the workflow. This interoperability is possible due to the use of Openapi specification and can be achieved at various levels.

Matching the name and content of a JSON schema (Level 0)

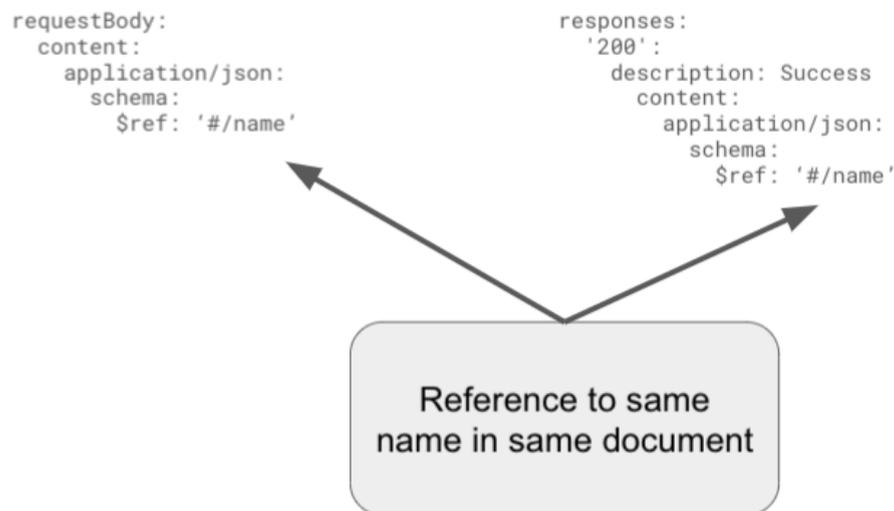


figure 4.1 basic json matching

Matching the form of the JSON file (Level 1) allowing for independent schemas to be interoperable:

```

name:
  type: object
  properties:
    partOfSpeech:
      type: string

```

```

differentName:
  type: object
  properties:
    partOfSpeech:
      type: string

```

Schema contain same property set

figure 4.2 basic json matching

Higher levels of interoperability are achieved by means of using the RDF representation of data (Level 2) to match schemes that are not naturally compatible and the use of the linking tools developed in Task 3.2 (Level 3) and the transformation tools in Task 3.1 (Level 4).

```

name:
  type: object
  properties:
    partOfSpeech:
      type: string

```

```

differentName:
  type: object
  properties:
    pos:
      type: string

```

```

{
  "partOfSpeech":
  "http://www.lexinfo.net/ontology/3.0/partOfSpeech"
}

```

```

{
  "pos":
  "http://www.lexinfo.net/ontology/3.0/partOfSpeech"
}

```

Services can be matched because JSON-LD context uses same property

figure 4.3 rdf matching

5. Usage

In this section we list the basic usage of Teanga as available in the official documentation in : <https://pret-a-llod.github.io/teanga/>

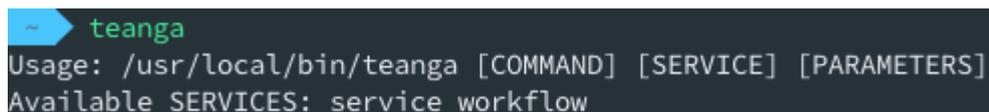
Installation

On your terminal run the following command :

```
curl https://raw.githubusercontent.com/Pret-a-LLOD/teanga/master/CLI/install.sh | sudo bash
```

It's a very small tool, so it should take a few seconds to finish.

It will install the CLI Teanga, the user can verify the installation by typing "teanga".



```
teanga
Usage: /usr/local/bin/teanga [COMMAND] [SERVICE] [PARAMETERS]
Available SERVICES: service workflow
```

Figure 5.1: Example of using teanga command

How to run a Workflow

1. Download the workflow example or Create your own workflow file following how to create a workflow in the Teanga documentation
2. Run Workflow:

```
teanga run workflow -f ./workflows/my_teanga_workflow.json -p 8080
```

The user can verify the workflow run by accessing localhost:8080

How to create a workflow in Teanga

The creation of workflows in Teanga is very generic and flexible. Through the UI the user can select services and add dependencies among services. Currently Teanga does not help the users with their workflows, meaning that the users need to have an use case in mind before implementing it.

Integrating new Services in Teanga

Services in Teanga must be a REST API that returns JSON-LD outputs. You can find a detailed guideline in the documentation.

There are 4 essential steps to make a rest api to work in Teanga:

1. Creating a Rest API app

- Create an application that receives http requests
- Expose your application in some port (8080 is recommended)
- Return a valid json format as output

2. Describing the Rest API with OpenAPI specification

- Inform the port of the application in the info section of the file

- Describe all your endpoints in the paths section of the file

3. Dockerizing your api

- Copy the openapi.yaml file to the root of the docker container “/openapi.yaml”

4. Publishing the Docker Image on Docker Hub

- create an account in Docker Hub
- upload docker image

Conclusion

Teanga is a workflow manager that helps users create, explore and execute complex workflows at the same time enables service providers to reliably integrate and test their service in it. This is possible by leveraging the power of the Semantic Web by the use of RDF technologies and matching strategies to connect services and also solid technologies and standards. The main technologies used were Apache Airflow, Docker, rest apis, openapi specification which were successfully tied together as an architecture. Our choices showed to be accessible with an easy CLI installation and usable with some working workflow study cases such as DKPRO tagger (example in the Teanga documentation). Currently, the main challenge in Teanga is executing large and processing heavy workflows. As such, Teanga is a flexible tool for the development of NLP pipelines and we will further expand on the scalability of the system in D3.6.



References

Open API Specifications - <https://oai.github.io/>

Apache Airflow - <https://airflow.apache.org/>

Docker - <https://docs.docker.com/get-started/overview/>

REST APIs - <https://www.redhat.com/en/topics/api/what-is-a-rest-api>

Teanga Documentation - <https://pret-a-llod.github.io/teanga/>

