



Universidad Politécnica  
de Madrid



**Escuela Técnica Superior de  
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

**Creación de una Web demostradora  
sobre Construcción de APIs REST a  
partir de Ontologías y Grafos de  
Conocimientos**

Autor: David Menéndez-Morán Fuentes

Tutor(a): Oscar Corcho, Cotutores: Daniel Garijo, Paola  
Espinoza

Madrid, <<Mayo 2021>>

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

*Trabajo Fin de Grado*

*Grado en Ingeniería Informática*

*Título: Creación de una Web demostradora sobre Construcción de APIs REST a partir de Ontologías y Grafos de Conocimientos*

*Mes: Mayo Año: 2021*

*Autor:* David Menéndez-Morán Fuentes

*Tutor:*

Oscar Corcho

Inteligencia Artificial

ETSI Informáticos

Universidad Politécnica de Madrid

# Resumen

En este trabajo se van a utilizar, probar y analizar bajo una métrica unas herramientas capaces de crear Webs API con un gráfico de conocimiento como base, estas están descritas con más detalle en el artículo de Paola Espinoza [1].

En concreto para cada herramienta se va a evaluar cuáles son sus pasos de ejecución con su configuración inicial e indicar que pasos son necesarios para cambiar dicha configuración. Además, se indicarán otros aspectos como el estado de su documentación, si la herramienta se mantiene activa, es decir, recibe mantenimiento por parte de los autores y describir la sensación general de la herramienta, lo fácil o difícil que ha sido utilizarla.

# **Abstract**

In this project some tools capable of creating a Web API with a Knowledge Graph as a base, are going to be used, tested and analysed with a metric in mind. The tools covered in this project are described with more detail in Paola Espinoza paper [1].

For each tool, it is going to be evaluated which are the required steps to execute the initial configuration and document which steps are needed to change the configuration. Also, other aspects will be noted as well, like the state of the tool's documentation, if such tool has support by the authors and describe the general sensation using the tool, if it has been easy or difficult to use.

# Tabla de contenidos

<b>1</b>	<b>Introducción.....</b>	<b>1</b>
<b>2</b>	<b>Estado del Arte .....</b>	<b>3</b>
<b>3</b>	<b>Desarrollo .....</b>	<b>5</b>
3.1	ELDA.....	5
3.1.1	Configuración .....	5
3.1.2	Ejecución .....	6
3.1.3	Ejemplos .....	6
3.1.4	Puntos débiles.....	8
3.1.5	Propuestas de Mejora.....	8
3.2	LODI/RDF-Explorer .....	8
3.2.1	Configuración .....	8
3.2.2	Ejecución .....	9
3.2.3	Ejemplos .....	9
3.2.4	Puntos débiles.....	10
3.2.5	Propuestas de Mejora.....	10
3.3	GRLC .....	10
3.3.1	Configuración .....	10
3.3.2	Ejecución .....	11
3.3.3	Ejemplos .....	12
3.3.4	Puntos débiles.....	16
3.3.5	Propuestas de Mejora.....	16
3.4	Apache Marmotta.....	17
3.4.1	Configuración .....	17
3.4.2	Ejecución .....	17
3.4.3	Ejemplos .....	18
3.4.4	Puntos débiles.....	21
3.4.5	Propuestas de Mejora.....	21
3.5	RAMOSE .....	21
3.5.1	Configuración .....	21
3.5.2	Ejecución .....	22
3.5.3	Ejemplos .....	22
3.5.4	Puntos débiles.....	26
3.5.5	Propuestas de Mejora.....	26
3.6	WALDER .....	27
3.6.1	Configuración .....	27
3.6.2	Ejecución .....	28
3.6.3	Ejemplos .....	28
3.6.4	Puntos débiles.....	30

3.6.5	Propuestas de Mejora.....	30
<b>4</b>	<b>Resultados y conclusiones .....</b>	<b>32</b>
<b>5</b>	<b>Análisis de Impacto .....</b>	<b>35</b>
<b>6</b>	<b>Bibliografía .....</b>	<b>36</b>

# 1 Introducción

Los grafos de conocimiento se han convertido en una parte imprescindible en las estructuras de datos que se utilizan actualmente, para ser utilizados en beneficio de las empresas que los usen, ya sea para hacer análisis económicos, mejorar la capacidad de búsqueda de los navegadores, etc.

De esta manera las empresas que utilizan dichas estructuras tienen una ventaja enorme debido a la gran cantidad de datos que tienen estos grafos de conocimiento.

Pero utilizar de manera adecuada estos grafos de conocimientos no es una tarea fácil, ya que los desarrolladores de aplicaciones que desean consumir estos datos no tienen la misma información disponible que aquellos desarrolladores que diseñaron el grafo de conocimiento. Estos trabajan con ontologías que siguen unas metodologías muy bien definidas que afectan al diseño de esta misma. Tanto que pueden volverse complejas y los recursos utilizados en el desarrollo no suelen ser abiertos al público, por lo que los desarrolladores que quieran consumir estos grafos de conocimiento van a trabajar de nuevo sobre lo que el desarrollador de la ontología ya había trabajado. Esto se traduce en un esfuerzo doble por parte del desarrollador de aplicaciones que se topa con los mismos problemas que tuvo que enfrentarse el desarrollador de la ontología cuando diseñó esta.

Además, los desarrolladores de aplicaciones no suelen estar familiarizados con los estándares de Web Semántica como OWL [2] y SPARQL [3], sino que están más familiarizados con formatos de datos como JSON [4] y usar “Application Programming Interfaces” (APIs) para acceder a sus datos. Para facilitar la vida a estos desarrolladores, se han creado varias herramientas capaces de unir los dos lenguajes, es decir, a partir de una ontología poder crear un servicio RESTful APIs [5]. De esta manera, el desarrollador de aplicaciones puede tener un acceso de los datos más sencillo y estas herramientas también ayudan al desarrollador de la ontología a guiar en cómo se deben consumir sus grafos de conocimiento.

Sin embargo, estas herramientas no son siempre fáciles de utilizar o entender su funcionamiento, alguna de ellas está limitada por el software que se utilice, es decir, puede que no funcione de la misma manera para todos los usuarios. Por lo que en este trabajo se ha decidido ir un paso adelante, de manera que todos los usuarios que lo deseen puedan usar dichas herramientas. Para ello se va a utilizar la tecnología Docker [6], que permite crear un contenedor simulando las condiciones necesarias que necesite la herramienta sin importar el hardware o software del desarrollador que va a utilizar dichas herramientas.

Las herramientas que se han seleccionado para aplicar este proceso provienen de un análisis previo realizado por Paola Espinoza en su artículo “Crossing the Chasm Between Ontology Engineering and Application Development: A Survey” [1] en el cual analiza las diferencias entre las herramientas y cuáles son sus funcionalidades.

Además, en este trabajo a parte de crear el contenedor para las herramientas seleccionadas, se va a crear una pequeña documentación en la que se explicarán dos aspectos fundamentales. Uno la explicación de como instalar la herramienta y otro los pasos que se tienen que seguir para



cambiar su configuración, en este caso se va a indicar como debe cambiarse el endpoint de la herramienta, así como los pasos para ejecutar dicha configuración.

## 2 Estado del Arte

Todas las herramientas analizadas y probadas en este trabajo provienen del artículo de Paola Espinoza, mencionado en el punto anterior, donde se realizó un estudio previo de un conjunto de herramientas. En este estudio, se analizó exhaustivamente el modo de como estas herramientas consumen grafos de conocimiento y el modo en el que se genera la API, comparándolas unas con otras. Para este trabajo se han escogido un total de seis herramientas procedentes del artículo mencionado.

Siendo las escogidas las siguientes:

- Linked Open Data Inspector (LODI) [7] / RDF-Explorer [8]
- Epimorphics Linked Data API Implementation (ELDA) [9]
- Apache Marmotta [10]
- Git repository linked data API constructor (GRLC) [11]
- Restful API Manager Over SPARQL Endpoints (RAMOSE) [12]
- Walder [13]

Ahora se procederá a realizar una breve descripción sobre cada una de las herramientas escogidas.

- LODI / RDF-Explorer: LODI es un servidor de Linked Data que refleja tanto sus vistas como la negociación de recursos en un SPARQL endpoint mediante HTML. LODI en su creación fue inspirado por otra de las herramientas que contiene el artículo Pubby, pero LODI ofrece más capacidades a los desarrolladores tratando con datos geoespaciales y detectando y mostrando archivos de imágenes.  
Sin embargo, en el momento de probar esta herramienta, no se consiguió instalar ni utilizando el Dockerfile que se proporciona ni intentando realizar la instalación local, estos detalles se verán reflejados en la parte de desarrollo. Por lo que se procedió a utilizar otra herramienta con características similares, siendo esta RDF-Explorer. Siendo esta herramienta también un Frontend de Linked Data para SPARQL endpoints, teniendo como característica adicional la capacidad de evitar interacciones que no lleven a ningún lado.
- ELDA: Es una implementación en Java de Linked Data API que provee una forma de acceso configurable a datos RDF utilizando URLs de tipo RESTful que son traducidas a queries e inyectadas al SPARQL endpoint elegido. Es necesario escribir una configuración de la API en RDF que indique cómo deben traducirse estas URLs en queries. De las herramientas analizadas en este trabajo, es la única que se implementa con Java y genera un ejecutable .jar.
- Apache Marmotta: Es una plataforma abierta para Linked Data, con el objetivo de proporcionar una implementación base de Linked Data Platform capaz de ser utilizada, ampliada y desarrollada fácilmente por organizaciones que deseen publicar Linked Data o construir aplicaciones para Linked Data. Es de las primeras herramientas en implementar el soporte para “LDP Basic Containers” y poder escoger el tipo de contenido que se fuese a devolver. Por desgracia esta herramienta

se retiró en noviembre de 2020 y no volverá a tener soporte, aun así, los contenidos de la herramienta y sus enlaces de descarga permanecen abiertos por lo que se continua con el análisis de esta herramienta.

- GRLC: Es un servidor liviano, que no consume muchos recursos, obteniendo SPARQL queries almacenadas en un repositorio de GitHub, de manera local o listadas en una URL y traduce dichas queries a Linked Data Web APIs. Esto habilita el acceso universal a Linked Data, puesto que no es necesario tener conocimiento de SPARQL, sino que se accede a través de una web API. La herramienta necesita de líneas especiales para ejecutar las queries denominadas 'decorators'. Estos 'decorators' deben ser introducidos como comentario al principio del archivo que contenga la query. Tienen funcionalidades como indicar el endpoint que se va a utilizar, indicar la paginación de la consulta o simplemente indicar un breve resumen de la operación en la interfaz de usuario. En concreto para este trabajo se ha centrado la atención en un 'decorator' específico, siendo este denominado 'transform'. Es el encargado de especificar la estructura JSON que va a mostrar la consulta realizada, tiene como condición que el resultado devuelto tiene que ser del tipo application/json.
- RAMOSE: Es una aplicación que permite un desarrollo ágil y publicación de una RESTful APIs totalmente documentada que realiza queries sobre SPARQL endpoints. Tanto los endpoints elegidos, como la documentación y las queries que se realicen tienen que estar definidas en un archivo de configuración hash-format. Este archivo de configuración se debe crear siguiendo la sintaxis especificada en la documentación de la herramienta. Por el momento esta herramienta solamente puede tratar con datos de tipo JSON o csv.
- Walder: Es una herramienta que ofrece la capacidad de configurar una página web o Web API sobre un grafo de conocimiento. Para ello es necesario tener un archivo de configuración con extensión '.yaml' en el que se define la Api, los métodos que se definen con las queries especificadas en lenguaje GRAPHQL-LD, la ruta de cada método y los mensajes del servidor.

Para cumplir con el objetivo de este trabajo de tener cada una de las herramientas en un contenedor y así poder utilizarse con cualquier sistema operativo se va a utilizar la Aplicación Docker Desktop [1].

Docker Desktop es una aplicación disponible tanto para Mac como Windows que permite construir y compartir aplicaciones que hayan montado un contenedor. De esta manera se habilita a los desarrolladores a separar la aplicación de la infraestructura pudiendo entregar el software rápidamente y a todo el mundo.

Además, se reduce el espacio que ocupan las aplicaciones puesto que dichos contenedores solamente tienen lo suficiente para poder ejecutar la herramienta a la que se le ha hecho un contendor.

Docker Desktop contiene a su vez a Docker Engine, siendo este el código abierto encargado de crear y montar los contenedores de las aplicaciones, Docker CLI client, encargado de realizar la consola de comandos para las

aplicaciones con contenedor, Docker Compose, que se encarga de definir y arrancar los contenedores.

Para realizar un contenedor de una herramienta, se necesita crear un archivo denominado “Dockerfile” el cual obtiene una imagen como base que se encuentra en Docker Hub, una página web donde se encuentran todas las imágenes de Docker subidas. Una vez obtenida la imagen base, se realizan las operaciones necesarias para que la herramienta pueda ejecutarse, ya sea instalar un comando necesario para la herramienta o la instalación de alguna dependencia. Por último, se indica el puerto que se expone y trabajará la herramienta con él. De esta forma el Dockerfile tendrá una apariencia similar a:

```
FROM python

RUN apt update
RUN apt-get install python3-pip -y

COPY requirements.txt ./

RUN pip3 install -r requirements.txt

COPY . .

EXPOSE 8080
```

En este caso se ha cogido como base la imagen de Python, se actualiza e instala un comando que se necesita en la herramienta, obtiene un archivo de requisitos y se instalan dichos requisitos, para finalizar se expone el puerto que la herramienta va a utilizar.

En caso de que la herramienta no posea un Dockerfile, se ha intentado crear uno siguiendo el esquema mostrado, sino se utiliza el Dockerfile proporcionado por los creadores de la herramienta.

## **3 Desarrollo**

En este apartado se explica el funcionamiento y los pasos necesarios para configurar cada una de las herramientas analizadas

### **3.1 ELDA**

#### **3.1.1 Configuración**

Para esta herramienta no posee un Dockerfile y no se ha conseguido crear uno. Por ello esta herramienta se ha probado de manera local. La instalación de la herramienta necesita previamente tener instalado Java [14] y Apache Maven [15], que se usarán para compilar y arrancar la herramienta.

Una vez se tenga la herramienta descargada de su repositorio, para compilar se ejecuta el siguiente comando dentro del directorio de la herramienta:

```
mvn clean package -Dmaven.test.skip=true
```

En este caso se utiliza la opción de saltarse las pruebas realizadas por Maven, puesto que, al tener el sistema operativo con el idioma en español, estas pruebas dan error al comprobar una fecha. En la que se espera que se devuelva un “Mon” de “Monday” y recibe un “lun” de lunes. Otra opción que también se ha realizado es cambiar el idioma del sistema operativo al inglés, en ese caso la opción del comando se omite y se realizan las pruebas correctamente.

La documentación indica que, al ejecutarse la herramienta con el ejemplo que trae deberían funcionar los siguientes enlaces:

[-standalone/hello/games](#)

[-standalone/again/games](#)

[-standalone/tiny/doc/school](#)

[-standalone/mini/doc/school](#)

[-standalone/full/doc/school](#)

Sin embargo, el único que funciona es again/games, no se ha conseguido resolver el porqué de este problema con el estudio realizado a esta herramienta.

### 3.1.2 Ejecución

Tras la compilación se crea un .jar (elda-standalone-\$VERSION-exec-war.jar) en la carpeta ..\elda-master\elda-standalone\target

Donde \$VERSION indica la versión utilizada.

Lo único necesario para arrancar la herramienta, por defecto en el puerto 8080 es ejecutar el siguiente comando:

```
java -jar elda-standalone-2.0.1-war-exec.jar
```

En nuestro caso la versión compilada es la 2.0.1, en el caso de que no se pueda utilizar el puerto por defecto, a la sentencia mencionada hay que añadir esta opción:


```
java -jar elda-standalone-2.0.1-war-exec.jar -httpPort 7070
```

Ahora la herramienta se levanta en el puerto 7070.

### 3.1.3 Ejemplos

Por los problemas comentados en el apartado de Configuración, la pruebas solo se han realizado sobre el enlace de again/games, de tal manera que la ruta completa sería:

<http://localhost:8080/standalone/again/games>



go to ...
page
data format
view

# Linked data API

## A Brief History of the World

https://rawgit.com/epimorphics/elda/gh-pages/demo/vocabularies/games.ttl#BriefHistoryOfTheWorld

a Board Game

label	A Brief History of the World
players	3, 4, 5, 6
pub year	2009
designed by	Ragnar Brothers
games dataset	dataset
same as	brief history of the world1.html

## A Few Acres of Snow

https://rawgit.com/epimorphics/elda/gh-pages/demo/vocabularies/games.ttl#a-few-acres-of-snow

a Board Game

label	A Few Acres of Snow
play time minutes	60
players	2
pub year	2011
designed by	Martin Wallace
games dataset	dataset
same as	a few acres of snow 2

En el que se muestran una serie de juegos de mesa, con sus distintos datos, como el año de publicación, el número de jugadores posibles o el creador del juego.

La herramienta permite filtrar cualquiera de los campos mostrados mediante argumentos en la URL, sin embargo, la redirección no es completa y es necesario añadir “standalone” a la URL justo después de la indicación del puerto y antes de “again” separado por barras ‘/’.

A continuación, se va a mostrar los juegos que tengan como año de publicación 2009



# Linked data API

[go to ... ▾](#)
[page ▾](#)
[data format ▾](#)
[view ▾](#)

## A Brief History of the World

<https://rawgit.com/epimorphics/elda/gh-pages/demo/vocabularies/games.ttl#BriefHistoryOfTheWorld>  
 a Board Game

label	A Brief History of the World ▾
players	3 ▾, 4 ▾, 5 ▾, 6 ▾
pub year	2009 ▾
designed by	Ragnar Brothers ▾
games dataset	dataset ▾
same as	brief history of the world1.html ▾

## Last Train to Wensleydale

<https://rawgit.com/epimorphics/elda/gh-pages/demo/vocabularies/games.ttl#last-train-to-wensleydale>  
 a Board Game

label	Last Train to Wensleydale ▾
play time minutes	120 ▾
players	3 ▾, 4 ▾
pub year	2009 ▾
designed by	Martin Wallace ▾
games dataset	dataset ▾
same as	last train to wensleydale ▾

Mientras que en la consulta anterior se mostraban los juegos “A Brief History of the World” y “A Few Acres of Snow”, este último ya no aparece dado que su año de publicación era en 2011 y ahora aparece el juego “Last Train to Wensleydale”.

Por desgracia para esta herramienta no se ha conseguido que funcione con otros datos de un endpoint diferente, por lo que sólo se pueden realizar esos filtrados en ese ejemplo específico.

#### **3.1.4 Puntos débiles**

Cómo puntos débiles que destacar en la herramienta, la documentación, aunque extensa no acaba de explicar del todo el funcionamiento de la herramienta, se centra más en describir partes del código del ejemplo, los cuales no indica en que carpeta del proyecto se encuentran.

Además, la complejidad del ejemplo, por la gran cantidad de correlación que tiene las clases entre sí, hace que sea muy complicado el poder crear un ejemplo desde cero.

#### **3.1.5 Propuestas de Mejora**

Las propuestas de mejora recomendadas para esta herramienta serían arreglar o indicar cómo ha de usarse la herramienta dentro de la documentación, con ejemplos paso a paso. Además de especificar con más claridad que debe cambiar un desarrollador para elegir otro endpoint o conjunto de pruebas.

### **3.2 LODI/RDF-Explorer**

Al intentar probar LODI la configuración inicial del Dockerfile no funciona correctamente, tras montar el Dockerfile y ejecutarlo, la herramienta se queda a la espera, pero no llega a levantar ningún servicio. Y al probar la herramienta de forma local, se instalan las dependencias indicadas en la documentación, pero en el momento de ejecutar el código tiene errores de compilación que no se han conseguido solucionar. Por lo que se pasó a analizar RDF-Explorer una herramienta similar.

#### **3.2.1 Configuración**

Hay que indicar que los enlaces de la configuración inicial para el localhost no funcionan, debido a que ya no existen (error 404 HTTP).

Para añadir a la herramienta un SPARQL endpoint solo hay que añadir el endpoint en formato JSON dentro de la carpeta “config”, añadir dentro de dev.json el siguiente fragmento de código:

```
"endpoints": {  
  "dbpedia": {  
    "endpointQuery": " https://dbpedia.org/sparql",  
    "base": "http://dbpedia.org"  
  }  
}
```

En este caso se ha añadido el endpoint de dbpedia donde se realizarán las queries de prueba.

### 3.2.2 Ejecución

Se descarga la herramienta desde GitHub desde este enlace:

[https://github.com/KnowledgeCaptureAndDiscovery/rdf\\_explorer](https://github.com/KnowledgeCaptureAndDiscovery/rdf_explorer)

La herramienta posee un archivo Dockerfile, para arrancar el servicio se construye utilizando:

```
docker-compose build
```

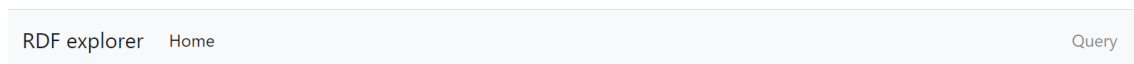
Una vez compilado ejecutar con:

```
docker-compose up -d
```

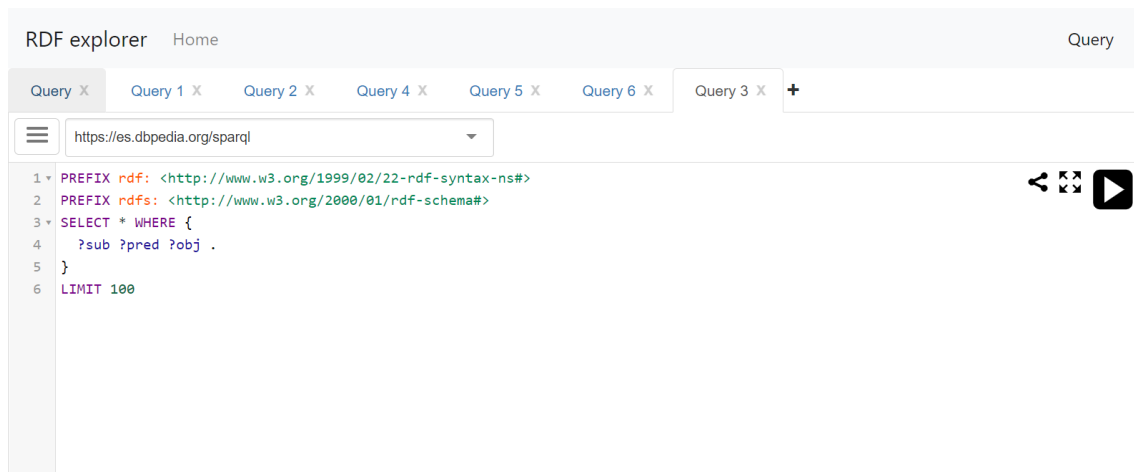
El servicio por defecto se encuentra en <http://localhost:7070>

### 3.2.3 Ejemplos

Una vez ejecutado los comandos mencionados anteriormente, se habrá creado una instancia de RDF-Explorer en Docker Desktop y se puede acceder al servicio mediante el puerto mencionado:



A continuación, se procede a la parte de las queries haciendo click en la esquina superior derecha.



Ahora se elige el SPARQL endpoint al que se quiera realizar la consulta, en esta prueba se utiliza el endpoint de dbpedia, y se indica la query que se va a ejecutar.



<div> <div>Table</div> <div>Response</div> <div>Pivot Table</div> <div>Google Chart</div> <div>Geo</div> <div>Download</div> <div>Code</div> </div>			
Showing 1 to 50 of 100 entries (in 0.062 seconds)		<div>Search:</div> <div></div> <div>Show 50 entries</div>	
sub	pred	obj	
1 <a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property">http://www.w3.org/1999/02/22-rdf-syntax-ns#Property</a>	
2 <a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property">http://www.w3.org/1999/02/22-rdf-syntax-ns#Property</a>	
3 <a href="http://www.w3.org/2001/XMLSchema#anyURI">http://www.w3.org/2001/XMLSchema#anyURI</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Datatype">http://www.w3.org/2000/01/rdf-schema#Datatype</a>	
4 <a href="http://www.w3.org/2001/XMLSchema#boolean">http://www.w3.org/2001/XMLSchema#boolean</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Datatype">http://www.w3.org/2000/01/rdf-schema#Datatype</a>	
5 <a href="http://www.w3.org/2001/XMLSchema#date">http://www.w3.org/2001/XMLSchema#date</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Datatype">http://www.w3.org/2000/01/rdf-schema#Datatype</a>	
6 <a href="http://www.w3.org/2001/XMLSchema#dateTime">http://www.w3.org/2001/XMLSchema#dateTime</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Datatype">http://www.w3.org/2000/01/rdf-schema#Datatype</a>	
7 <a href="http://www.w3.org/2001/XMLSchema#dateTime">http://www.w3.org/2001/XMLSchema#dateTime</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Datatype">http://www.w3.org/2000/01/rdf-schema#Datatype</a>	
8 <a href="http://www.w3.org/2001/XMLSchema#double">http://www.w3.org/2001/XMLSchema#double</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Datatype">http://www.w3.org/2000/01/rdf-schema#Datatype</a>	
9 <a href="http://www.w3.org/2001/XMLSchema#float">http://www.w3.org/2001/XMLSchema#float</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Datatype">http://www.w3.org/2000/01/rdf-schema#Datatype</a>	
10 <a href="http://www.w3.org/2001/XMLSchema#gDay">http://www.w3.org/2001/XMLSchema#gDay</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Datatype">http://www.w3.org/2000/01/rdf-schema#Datatype</a>	
11 <a href="http://www.w3.org/2001/XMLSchema#gMonth">http://www.w3.org/2001/XMLSchema#gMonth</a>	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#type">http://www.w3.org/1999/02/22-rdf-syntax-ns#type</a>	<a href="http://www.w3.org/2000/01/rdf-schema#Datatype">http://www.w3.org/2000/01/rdf-schema#Datatype</a>	

Y estos son los resultados de la query que se ha ejecutado, mostrando la tripleta de datos que se habían pedido.

### 3.2.4 Puntos débiles

La herramienta tiene en su configuración inicial una gran cantidad de enlaces caídos o enlaces que no tienen ningún dato. La documentación en general es muy escasa, tanto de la configuración de la propia herramienta o las posibilidades que esta ofrece. La parte de documentación para añadir un nuevo endpoint es ambigua y escasa. Siendo esta una parte importante de la documentación no deja claro si hay que crear un nuevo archivo añadiendo las líneas que se indican o insertar dichas líneas dentro de un archivo de configuración existente.

### 3.2.5 Propuestas de Mejora

Como punto principal a mejorar, sería realizar una documentación detallada sobre su uso y configuración.

Añadir ejemplos paso a paso de como configurar la herramienta eliminando la ambigüedad mencionada en el punto anterior, además de añadir imágenes de que debería visualizar la herramienta cuando se ha configurado correctamente.

## 3.3 GRQL

### 3.3.1 Configuración

Para que la herramienta funcione, necesita acceder a un repositorio de GitHub que sea público. En dicho repositorio solo lee los archivos que tengan como extensión ‘.rq’ o ‘.sparql’, de esta manera identifica que es una SPARQL query.

Para la creación de estas queries se necesitan definir todos los prefijos necesarios, además de añadir un decorator indicando el endpoint al principio del archivo de esta manera:

```
#+endpoint: https://dbpedia.org/sparql
```

Cambiando el sparql endpoint al que se quiera realizar la consulta.

Además de como cambiar un endpoint para esta herramienta, se pidió analizar el uso de otro decorator, siendo este transform, que tiene la siguiente estructura:

```
#+ transform: {  
#+   "key": "?p",  
#+   "value": "?o",  
#+   "$anchor": "key"  
#+ }
```

Utilizando este decorator, la herramienta es capaz de visualizar los datos en JSON de la manera en la que se indique dentro del decorator, pudiendo realizar un esquema de los datos mucho más visibles y concisos. Para que la sentencia transform tenga efecto, se tiene que elegir que el servicio nos devuelva un JSON obligatoriamente, si no el servidor nos devuelve un error.

Este es un ejemplo de query que ha de crearse en el repositorio con alguna de las extensiones mencionadas:

```
#+ endpoint: https://dbpedia.org/sparql  
#+ transform: {  
#+   "band": "?band",  
#+   "album": "?album",  
#+   "$anchor": "band"  
#+ }  
  
PREFIX dbo: <http://dbpedia.org/ontology/>  
PREFIX dbp: <http://dbpedia.org/property/>  
PREFIX schema: <http://schema.org/>  
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  
  
SELECT ?band ?album ?genre WHERE {  
  ?band rdf:type dbo:Band .  
  ?album rdf:type schema:MusicAlbum .  
  ?band dbo:genre ?genre .  
  ?album dbp:artist ?band .  
}  
LIMIT 100
```

### 3.3.2 Ejecución

La herramienta está dockerizada, para arrancar se ejecuta el siguiente comando:

```
docker run -it --rm -p 8088:80 -e clariah/grlc
```

El Docker permite opciones como cambiar el nombre del servidor, indicar un token de acceso GitHub o establecer un endpoint fijo para todas las queries si se desea.

Para ejecutar de manera local y para que funcione correctamente la instancia de swagger que se genera, es necesario indicarle un nombre de servidor y un GitHub Access token al ejecutar la herramienta.

```
docker run -it --rm -p 8088:80 -e GRPC_SERVER_NAME=grpc.io -e
GRPC_GITHUB_ACCESS_TOKEN=xxx clariah/grpc
```

De esta manera estamos nombrando al servidor “grpc.io” y donde están las xxx hay que indicar el token de acceso de GitHub. Este se genera desde GitHub y el token debe tener los permisos para acceder a repositorios públicos y poder realizar obtener datos de los repositorios. El token que se ha utilizado para realizar las pruebas es “ghp\_KXAlqIgMPOACzl0MtlyWfqDcpVjUXf3WKvr4”, por lo que una ejecución válida es la siguiente:


```
docker run -it --rm -p 8088:80 -e GRPC_SERVER_NAME=grpc.io -e
GRPC_GITHUB_ACCESS_TOKEN=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
clariah/grpc
```

Una vez arrancado se muestra una pantalla informativa y hay que redirigirse a <http://grpc.io/api/username/repo/>. Para las pruebas utilizadas se ha utilizado:

<http://localhost:8088/api/david-menendez/grpc-queries/>

### 3.3.3 Ejemplos

En el repositorio mencionado tiene tres casos de prueba, siendo estos distintas queries con distintos endpoints. GRPC crea una instancia de swagger y en este caso se vería así:

 grlc

# grlc-queries

0e69cfdbc03ac59d30eb9533291d4b1aa58bca84

[ Base URL: [grlc.io/api-git/david-menendez/grlc-queries/](http://grlc.io/api-git/david-menendez/grlc-queries/) ]  
[/api/david-menendez/grlc-queries/swagger](http://api/david-menendez/grlc-queries/swagger)

queries for grlc  
[david-menendez - Website](#)  
License

default

GET /ciudades

GET /dbpediaquery

GET /dbpediaquerySin

Models

Message >

Como ejemplo se va a ejecutar la sentencia dbpediaquery, al realizar click en dicha sentencia se muestra la query que se va a ejecutar.

GET

/dbpediaquery

```
#+ transform: {
#+   "band": "?band",
#+   "album": "?album",
#+   "$anchor": "band"
#+ }

PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX schema: <http://schema.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?band ?album ?genre WHERE {
  ?band rdf:type dbo:Band .
  ?album rdf:type schema:MusicAlbum .
  ?band dbo:genre ?genre .
  ?album dbp:artist ?band .
} LIMIT 100
```

Parameters

Try it out

Name	Description
endpoint	Alternative endpoint for SPARQL query
string (query)	Default value: <a href="https://dbpedia.org/sparql">https://dbpedia.org/sparql</a>

Responses

Response content type

application/json

Esta es la prueba en la que se va a pedir un JSON con un formato específico, en este caso sólo se mostrarán la banda y sus álbumes, aunque la query contenga más datos, el decorator fuerza a que tomen la forma que se les indique.

Se procede a realizar la prueba haciendo click en “Try it out” y nos devolverá lo indicado en “Response content type” en este caso y dado que es la sentencia que contiene el decorator transform se necesita JSON.

Server response

CodeDetails

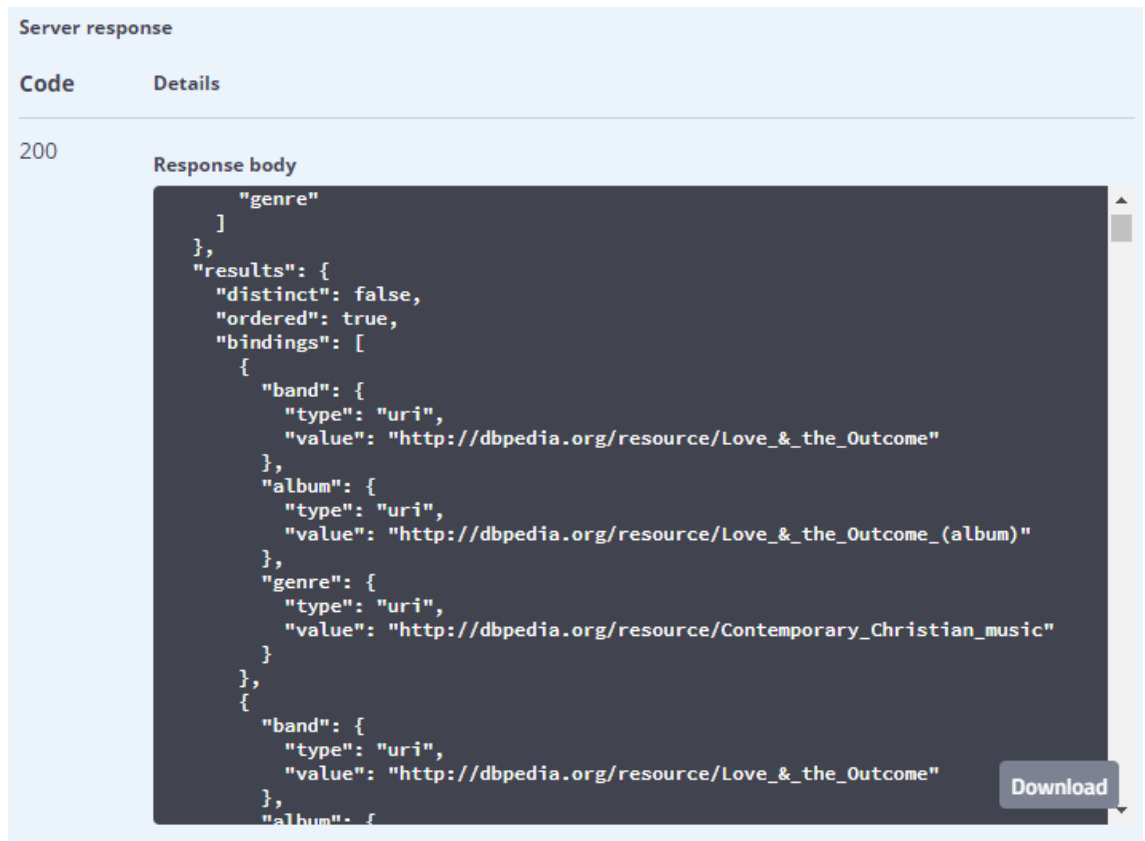
200

Response body

```
"album": "http://dbpedia.org/resource/Love_(Love_album)",
"band": "http://dbpedia.org/resource/Love_(band)"
},
{
  "album": "http://dbpedia.org/resource/Love_Amongst_Ruin_(album)",
  "band": "http://dbpedia.org/resource/Love_Amongst_Ruin"
},
{
  "album": [
    "http://dbpedia.org/resource/Lovelinus",
    "http://dbpedia.org/resource/LoveLyz8"
  ],
  "band": "http://dbpedia.org/resource/LoveLyz"
},
{
  "album": [
    "http://dbpedia.org/resource/Loverboy_(Loverboy_album)",
    "http://dbpedia.org/resource/Loverboy_Classics",
    "http://dbpedia.org/resource/Lovin'_Every_Minute_of_It"
  ],
  "band": "http://dbpedia.org/resource/Loverboy"
},
{
  "album":
    "http://dbpedia.org/resource/Loved_by_Thousands,_Hated_by_Millions",
  "band": "http://dbpedia.org/resource/M.O.D."
}
```

Download

Y el servidor nos devuelve los datos con el formato que hemos elegido dentro del transform. Para ver la diferencia y el efecto que tiene este decorator en la visualización de los datos, se procede a ejecutar la query denominada dbpediaquerySin, siendo la misma sentencia que acabamos de ejecutar, pero sin el decorator transform.



Este es el resultado JSON que devuelve el servicio ejecutando dicha query. Sin el decorator los datos mostrados son mucho más difíciles de ver y concretar que es cada línea y a quien pertenecen. Sin embargo, esta sentencia devuelve todos los datos que se realizaron en la query, que no se visualizaron en la sentencia anterior como es el género. Dado que no tiene ningún decorator indicando que datos debe mostrar y como mostrar dichos datos, visualiza todos los datos que se pidieron en la query.

### 3.3.4 Puntos débiles

No tiene puntos débiles mencionables con el estudio realizado, debido a que, de todas las herramientas analizadas, GRLC es de las que más soporte continuo recibe y sobre todo por su alta y buena calidad de documentación que posee. Prácticamente para todo lo que ofrece la herramienta tiene ejemplos y puntos paso a paso fáciles de seguir en cómo debe utilizarse.

### 3.3.5 Propuestas de Mejora

De la parte analizada sobre los decorator de esta herramienta, transform el más reciente añadido, no posee el mismo detalle y cantidad de documentación que el resto de la herramienta. Por ejemplo, no viene definido que función realiza el operador “\$anchor” en el decorator

transform. Por lo que añadir esta documentación o un enlace sería una buena mejora.

## 3.4 Apache Marmotta

### 3.4.1 Configuración

Esta herramienta tiene un funcionamiento completamente distinto a las demás herramientas analizadas en el trabajo. Debido a que se parece más a un explorador de archivos, en vez de a un generador de APIs mediante un grafo de conocimientos. Como ya se ha comentado, esta herramienta es la única que a sido retirada y no volverá a tener soporte.

La herramienta cuenta con una imagen en Docker la cual se ha utilizado para realizar las pruebas correspondientes, lo primero es obtener dicha imagen con el comando:

```
docker pull apache/marmotta
```

Cuando se levante el servicio en la fase de ejecución, el servidor se encuentra en la siguiente dirección:

<http://localhost:8080/marmotta>

Ahora se necesita un archivo '.ttl' el cual se va a añadir a la herramienta y se podrá redirigir hasta su localización, en este caso, para comprobar el correcto funcionamiento de la herramienta se han utilizado una batería de prueba procedentes de: <https://www.w3.org/TeamSubmission/turtle/>

Tras tener la batería de pruebas descargada, es necesario añadir el archivo a la herramienta, este comando ha de ejecutarse una vez el servicio esté arrancado:

```
curl -i -X POST -d @test-30.ttl -H "Content-Type: text/turtle" -H "Slug: test" http://localhost:8080/marmotta/ldp
```

Con este comando se está añadiendo a la herramienta el archivo test-30.ttl a la dirección indicada, ahora solo queda hacer que el servicio acepte esta petición e indique la subcarpetas en la que se encuentra, utilizando el siguiente comando:

```
curl -i -H "Accept: text/turtle" http://localhost:8080/marmotta/ldp/test
```

En esta sentencia se acepta el archivo enviado en la subcarpetas test.

### 3.4.2 Ejecución

Lo único necesario para arrancar la herramienta tras haber obtenido su imagen es ejecutar el siguiente comando:

```
docker run -p 8080:8080 apache/marmotta
```

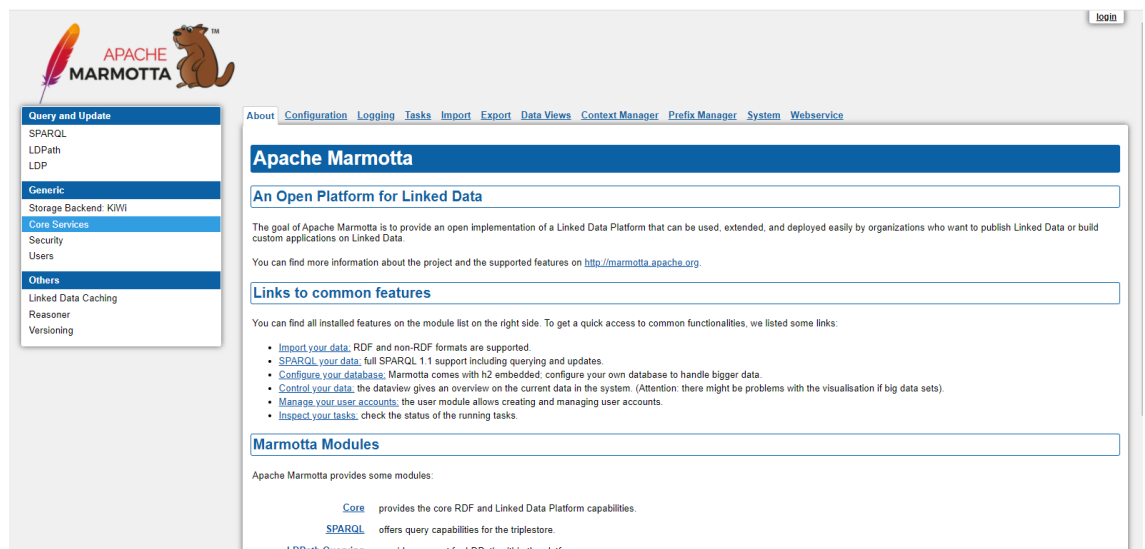


Con esta sentencia, se levanta el servicio con el puerto 8080, que se ha utilizado para la realización de los ejemplos. Se puede elegir otro puerto si este ya está en uso sustituyéndolo en la sentencia anterior, pero hay que tener en cuenta que el puerto del localhost cambiaría al nuevo puerto escogido. En este caso tras levantar el servicio con la sentencia mostrada, la herramienta se encuentra en:

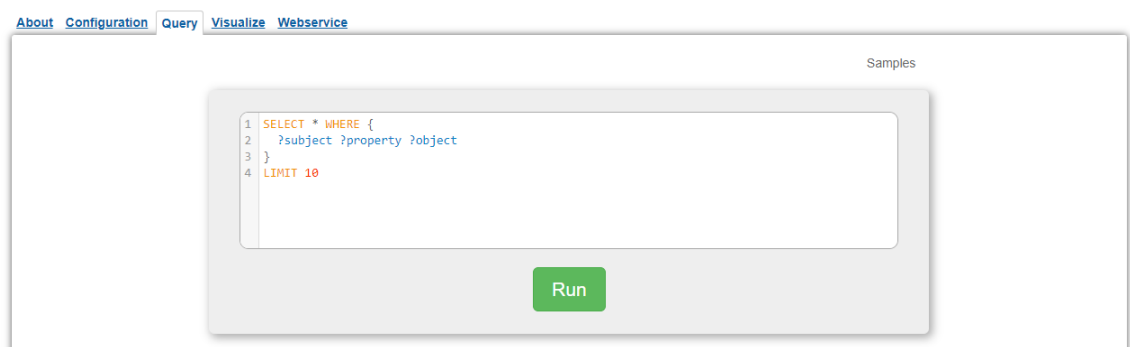
<http://localhost:8080/marmotta>

### 3.4.3 Ejemplos

Nada más acceder a la dirección indicada, la herramienta nos muestra su menú:



Para navegar por los recursos que se han introducido, hay que entrar en el apartado de “SPARQL” situado en el menú de la izquierda de la pantalla, una vez en ese menú, se va a la parte de “query”.



Que nos muestra la query ejecutada y sus resultados:

Browse	XML	JSON	CSV
--------	-----	------	-----

< pref
Rows 1 to 10 of overall 10
next >

subject	property	object
<a href="http://localhost:8080/marmotta/ldp">http://localhost:8080/marmotta/ldp</a>	<a href="#">rdfs:label</a>	Marmotta's LDP Root Container
<a href="http://localhost:8080/marmotta/ldp">http://localhost:8080/marmotta/ldp</a>	<a href="#">rdf:type</a>	<a href="http://www.w3.org/ns/ldp#Resource">http://www.w3.org/ns/ldp#Resource</a>
<a href="http://localhost:8080/marmotta/ldp">http://localhost:8080/marmotta/ldp</a>	<a href="#">rdf:type</a>	<a href="http://www.w3.org/ns/ldp#RDFSSource">http://www.w3.org/ns/ldp#RDFSSource</a>
<a href="http://localhost:8080/marmotta/ldp">http://localhost:8080/marmotta/ldp</a>	<a href="#">rdf:type</a>	<a href="http://www.w3.org/ns/ldp#Container">http://www.w3.org/ns/ldp#Container</a>
<a href="http://localhost:8080/marmotta/ldp">http://localhost:8080/marmotta/ldp</a>	<a href="#">rdf:type</a>	<a href="http://www.w3.org/ns/ldp#BasicContainer">http://www.w3.org/ns/ldp#BasicContainer</a>
<a href="http://localhost:8080/marmotta/ldp">http://localhost:8080/marmotta/ldp</a>	<a href="http://www.w3.org/ns/ldp#interactionModel">http://www.w3.org/ns/ldp#interactionModel</a>	<a href="http://www.w3.org/ns/ldp#Container">http://www.w3.org/ns/ldp#Container</a>
<a href="http://localhost:8080/marmotta/ldp">http://localhost:8080/marmotta/ldp</a>	<a href="#">dct:created</a>	2021-06-03T08:20:24.000Z <a href="#">^^xsd:dateTime</a>
<a href="http://www.w3.org/ns/ldp#RDFSSource">http://www.w3.org/ns/ldp#RDFSSource</a>	<a href="#">rdf:type</a>	<a href="#">rdfs:Class</a>
<a href="http://www.w3.org/ns/ldp#RDFSSource">http://www.w3.org/ns/ldp#RDFSSource</a>	<a href="#">rdfs:comment</a>	A Linked Data Platform Resource (LDPR) whose state is represented as RDF.
<a href="http://www.w3.org/ns/ldp#RDFSSource">http://www.w3.org/ns/ldp#RDFSSource</a>	<a href="#">rdfs:isDefinedBy</a>	<a href="http://www.w3.org/ns/ldp#">http://www.w3.org/ns/ldp#</a>

Para navegar hacia el contenedor recién añadido, hay que ir a al enlace de la columna 'object' de la derecha, el enlace de 'Container'.

Siendo esta la query que ejecuta:

[About](#)
[Configuration](#)
[Query](#)
[Visualize](#)
[Webservice](#)

Samples

```

1 SELECT DISTINCT ?property ?hasValue ?isValueOf WHERE {
2   { <http://www.w3.org/ns/ldp#Container> ?property ?hasValue }
3   UNION
4   { ?isValueOf ?property <http://www.w3.org/ns/ldp#Container> }
5 }
6 ORDER BY ?property ?hasValue ?isValueOf
7 LIMIT 1000

```

Run

Y los resultados de dicha query:

Browse	XML	JSON	CSV
--------	-----	------	-----

< pref

Rows 1 to 10 of overall 12

next >

property	hasValue	isValueOf
<a href="#">rdf:type</a>	<a href="#">rdfs:Class</a>	
<a href="#">rdf:type</a>		<a href="http://localhost:8080/marmotta/ldp">http://localhost:8080/marmotta/ldp</a>
<a href="#">rdf:type</a>		<a href="http://localhost:8080/marmotta/ldp/test">http://localhost:8080/marmotta/ldp/test</a>
<a href="#">rdf:type</a>		<a href="http://localhost:8080/marmotta/ldp/test-1">http://localhost:8080/marmotta/ldp/test-1</a>
<a href="#">rdfs:comment</a>	A Linked Data Platform RDF Source (LDP-RS) that also conforms to additional patterns and conventions for managing membership. Readers should refer to the specification defining this ontology for the list of behaviors associated with it.	
<a href="#">rdfs:isDefinedBy</a>	<a href="http://www.w3.org/ns/ldp#">http://www.w3.org/ns/ldp#</a>	
<a href="#">rdfs:label</a>	Container	
<a href="#">rdfs:subClassOf</a>	<a href="http://www.w3.org/ns/ldp#RDFSSource">http://www.w3.org/ns/ldp#RDFSSource</a>	
<a href="http://www.w3.org/2003/06/sw-vocab-status/ns#term_status">http://www.w3.org/2003/06/sw-vocab-status/ns#term_status</a>	stable	
<a href="http://www.w3.org/ns/ldp#interactionModel">http://www.w3.org/ns/ldp#interactionModel</a>		<a href="http://localhost:8080/marmotta/ldp">http://localhost:8080/marmotta/ldp</a>

Para este ejemplo, se añadió otro archivo ‘.ttl’ con las mismas instrucciones mencionadas, por lo que se puede ver como la herramienta diferencia su ruta, siendo una ../ldp/test y la otra ../ldp/test-1.

Para finalizar el ejemplo veremos el contenido que produce hacer click en la primera ruta mencionada:

[About](#)
[Configuration](#)
[Query](#)
[Visualize](#)
[Webservice](#)

Samples

```

1 SELECT DISTINCT ?property ?hasValue ?isValueOf WHERE {
2   { <http://localhost:8080/marmotta/ldp/test> ?property ?hasValue }
3   UNION
4   { ?isValueOf ?property <http://localhost:8080/marmotta/ldp/test> }
5 }
6 ORDER BY ?property ?hasValue ?isValueOf
7 LIMIT 1000

```

Run

Siendo esta la query que ejecuta.

Browse	XML	JSON	CSV
--------	-----	------	-----

< pref

Rows 1 to 8 of overall 8

next >

property	hasValue	isValueOf
<a href="#">dct:created</a>	2021-06-03T08:23:03.000Z <a href="#">^^xsd:dateTime</a>	
<a href="#">dct:modified</a>	2021-06-03T08:23:03.000Z <a href="#">^^xsd:dateTime</a>	
<a href="#">rdf:type</a>	<a href="#">http://www.w3.org/ns/ldp#BasicContainer</a>	
<a href="#">rdf:type</a>	<a href="#">http://www.w3.org/ns/ldp#Container</a>	
<a href="#">rdf:type</a>	<a href="#">http://www.w3.org/ns/ldp#RDFSSource</a>	
<a href="#">rdf:type</a>	<a href="#">http://www.w3.org/ns/ldp#Resource</a>	
<a href="#">http://www.w3.org/ns/ldp#contains</a>		<a href="#">http://localhost:8080/marmotta/ldp</a>
<a href="#">http://www.w3.org/ns/ldp#interactionModel</a>	<a href="#">http://www.w3.org/ns/ldp#Container</a>	

Junto con su contenido.

### 3.4.4 Puntos débiles

Cómo puntos débiles más destacables, sería que la documentación de cómo añadir un archivo no funciona correctamente, puesto que el enlace al que quiere añadir el archivo no existe. Y sobre todo la herramienta no llega a crear una Web API a través del archivo, sólo se puede navegar a través de estos.

### 3.4.5 Propuestas de Mejora

No puede tener propuestas de mejora puesto que esta herramienta ya ha sido descatalogada.

## 3.5 RAMOSE

### 3.5.1 Configuración

La herramienta utiliza archivos de configuración con la extensión ‘.hf’ donde se indican las propiedades de la api, indicando al endpoint al cual se van a realizar las consultas, la ruta por la cual se va a acceder a las distintas llamadas al igual que la propia consulta query. Para esta herramienta se ha creado el archivo de configuración “mi\_db\_test.hf” siendo este una versión simplificada de la configuración inicial “test.hf”.

Se ha creado un Dockerfile para esta herramienta, para poder ser ejecutada en cualquier máquina. Este Dockerfile tiene como base la imagen de Python y se le añaden las dependencias que necesita esta

herramienta como pip3 y la instalación del archivo “requirements.txt” para que la herramienta funcione con normalidad.

Para montar el Dockerfile utilizar la siguiente línea de comando en el directorio del Dockerfile:

```
Docker build -t <nombre> .
```

Donde <nombre> es el nombre que se le asignará a la imagen creada.

Una vez la imagen ha sido construida, utilizar Docker Desktop ir a la pestaña de “images” seleccionar la imagen creada y ejecutarla haciendo click en el botón “RUN”.

Una vez la imagen está en ejecución sólo queda abrir la herramienta a través del CLI creado por Docker Desktop.

### 3.5.2 Ejecución

Dentro del entorno CLI recién creado, para poder ejecutar la herramienta hay que entrar dentro de la carpeta ‘test’ y ejecutar el siguiente formato de instrucción:

```
python -m ramose -s <config.hf> -c '<route>'
```

Donde <config.hf> es el archivo de configuración, es decir, la definición de la api que se quiere utilizar y <'route'> es la ruta, definida dentro de la configuración, de cada método que exista en la api. Ejemplo de ejecución:

```
python -m ramose -s mi_db_test.hf -c '/api/v2/metadata'
```

La herramienta también permite levantar un servidor de pruebas utilizando la siguiente sentencia:

```
python -m ramose -s mi_db_test.hf -w 127.0.0.1:8080
```

Siendo la opción -w el puerto donde se quiera levantar el servicio, en este caso se pueden realizar las pruebas directamente en el navegador cambiando la URL de esta manera.

<http://localhost:8080/api/v2/metadata>

Sin embargo, esta funcionalidad no se ha conseguido reproducir con éxito en el Dockerfile, por lo que, si se desea utilizar el servicio de esta manera se tiene que instalar la herramienta de forma local.

### 3.5.3 Ejemplos

El CLI creado por el Dockerfile debe tener la siguiente estructura:

```
docker exec -it 61da73c7f6797b7a9bf0054834b24b299df718b5d0cd7128c3a87f16ca508b70 /bin/sh
# ls -l
total 192
-rwxr-xr-x 1 root root 155 Jun 1 09:28 Dockerfile
-rwxr-xr-x 1 root root 769 Mar 26 05:01 LICENSE
-rwxr-xr-x 1 root root 21072 Mar 26 05:01 README.md
drwxr-xr-x 2 root root 4096 May 31 19:21 __pycache__
drwxr-xr-x 1 root root 4096 May 12 01:56 bin
drwxr-xr-x 2 root root 4096 Mar 19 23:44 boot
drwxr-xr-x 5 root root 360 Jun 1 09:29 dev
drwxr-xr-x 1 root root 4096 Jun 1 09:29 etc
drwxr-xr-x 2 root root 4096 May 31 19:21 eval
drwxr-xr-x 2 root root 4096 Mar 19 23:44 home
drwxr-xr-x 1 root root 4096 May 12 01:57 lib
drwxr-xr-x 2 root root 4096 May 11 00:00 lib64
drwxr-xr-x 2 root root 4096 May 11 00:00 media
drwxr-xr-x 2 root root 4096 May 11 00:00 mnt
drwxr-xr-x 2 root root 4096 May 11 00:00 opt
dr-xr-xr-x 175 root root 0 Jun 1 09:29 proc
-rwxr-xr-x 1 root root 68017 Mar 26 05:01 ramose.py
-rwxr-xr-x 1 root root 238 Mar 26 05:01 requirements.txt
drwx----- 1 root root 4096 Jun 1 09:28 root
drwxr-xr-x 3 root root 4096 May 11 00:00 run
drwxr-xr-x 1 root root 4096 May 12 01:56/sbin
drwxr-xr-x 2 root root 4096 May 11 00:00/srv
dr-xr-xr-x 13 root root 0 Jun 1 09:29/sys
drwxr-xr-x 3 root root 4096 May 31 19:21/test
drwxrwxrwt 1 root root 4096 Jun 1 09:28/tmp
drwxr-xr-x 1 root root 4096 May 11 00:00/usr
drwxr-xr-x 1 root root 4096 May 11 00:00/var
#
```

A continuación, se procede como se ha mencionado a entrar en la carpeta “test” y realizar la siguiente línea de comando:

```
python -m ramose -s mi_db_test.hf -c '/api/v2/metadata'
```

Este método ejecuta esta query:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
```

```
PREFIX dbp: <http://dbpedia.org/property/>
```

```
PREFIX schema: <http://schema.org/>
```

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
SELECT ?band ?album ?genre WHERE {
    ?band rdf:type dbo:Band .
    ?album rdf:type schema:MusicAlbum .
    ?band dbo:genre ?genre .
    ?album dbp:artist ?band .
} LIMIT 100
```

```
docker exec -it 61da73c7f6797b7a9bf0054834b24b299df718b5d0cd7128c3a87f16ca508b70 /bin/sh
# cd test
# python -m ramose -s mi_db_test.hf -c '/api/v2/metadata'
# Response HTTP code: 200
# Body:
[
  {
    "band": "http://dbpedia.org/resource/Love_&_the_Outcome",
    "album": "http://dbpedia.org/resource/Love_&_the_Outcome_(album)",
    "genre": "http://dbpedia.org/resource/Contemporary_Christian_music"
  },
  {
    "band": "http://dbpedia.org/resource/Love_&_the_Outcome",
    "album": "http://dbpedia.org/resource/Love_&_the_Outcome_(album)",
    "genre": "http://dbpedia.org/resource/Pop_rock"
  },
  {
    "band": "http://dbpedia.org/resource/Love_(band)",
    "album": "http://dbpedia.org/resource/Love_(Love_album)",
    "genre": "http://dbpedia.org/resource/Psychedelic_pop"
  },
  {
    "band": "http://dbpedia.org/resource/Love_(band)",
    "album": "http://dbpedia.org/resource/Love_(Love_album)",
    "genre": "http://dbpedia.org/resource/Psychedelic_rock"
  },
  {
    "band": "http://dbpedia.org/resource/Love_(band)",
    "album": "http://dbpedia.org/resource/Love_(Love_album)",
    "genre": "http://dbpedia.org/resource/Folk_rock"
  }
]
```

Y este es el resultado de la consulta realizada, la herramienta por defecto devuelve un JSON. La herramienta solo admite csv y JSON, para que la herramienta devuelva la misma sentencia con un csv se debe ejecutar la siguiente sentencia:

```
python -m ramose -s mi_db_test.hf -c '/api/v2/metadata?format=csv'

docker exec -it 61da73c7f6797b7a9bf0054834b24b299df718b5d0cd7128c3a87f16ca508b70 /bin/sh
# python -m ramose -s mi_db_test.hf -c '/api/v2/metadata?format=csv'
# Response HTTP code: 200
# Body:
band,album,genre
http://dbpedia.org/resource/Love_&_the_Outcome,http://dbpedia.org/resource/Love_&_the_Outcome_(album),http://dbpedia.org/resource/Contemporary_Christian_music
http://dbpedia.org/resource/Love_&_the_Outcome,http://dbpedia.org/resource/Love_&_the_Outcome_(album),http://dbpedia.org/resource/Pop_rock
http://dbpedia.org/resource/Love_(band),http://dbpedia.org/resource/Love_(Love_album),http://dbpedia.org/resource/Psychedelic_pop
http://dbpedia.org/resource/Love_(band),http://dbpedia.org/resource/Love_(Love_album),http://dbpedia.org/resource/Psychedelic_rock
http://dbpedia.org/resource/Love_(band),http://dbpedia.org/resource/Love_(Love_album),http://dbpedia.org/resource/Folk_rock
http://dbpedia.org/resource/Love_(band),http://dbpedia.org/resource/Love_(Love_album),http://dbpedia.org/resource/Acid_rock
http://dbpedia.org/resource/Love_Amongst_Ruin,http://dbpedia.org/resource/Love_Amongst_Ruin_(album),http://dbpedia.org/resource/Hard_rock
http://dbpedia.org/resource/Love_Amongst_Ruin,http://dbpedia.org/resource/Love_Amongst_Ruin_(album),http://dbpedia.org/resource/Alternative_rock
http://dbpedia.org/resource/Lovelyz,http://dbpedia.org/resource/Lovelinus,http://dbpedia.org/resource/Bubblegum_pop
http://dbpedia.org/resource/Lovelyz,http://dbpedia.org/resource/Lovelinus,http://dbpedia.org/resource/Synth-pop
http://dbpedia.org/resource/Lovelyz,http://dbpedia.org/resource/Lovelinus,http://dbpedia.org/resource/K-pop
http://dbpedia.org/resource/Lovelyz,http://dbpedia.org/resource/Lovelyz8,http://dbpedia.org/resource/Bubblegum_pop
http://dbpedia.org/resource/Lovelyz,http://dbpedia.org/resource/Lovelyz8,http://dbpedia.org/resource/Synth-pop
http://dbpedia.org/resource/Lovelyz,http://dbpedia.org/resource/Lovelyz8,http://dbpedia.org/resource/K-pop
http://dbpedia.org/resource/Loverboy,http://dbpedia.org/resource/Loverboy_(Loverboy_album),http://dbpedia.org/resource/Hard_rock
http://dbpedia.org/resource/Loverboy,http://dbpedia.org/resource/Loverboy_(Loverboy_album),http://dbpedia.org/resource/Rock_music
```

Este es el resultado en formato csv.

La configuración elegida dispone de otro método al cual se le llama de la siguiente forma:

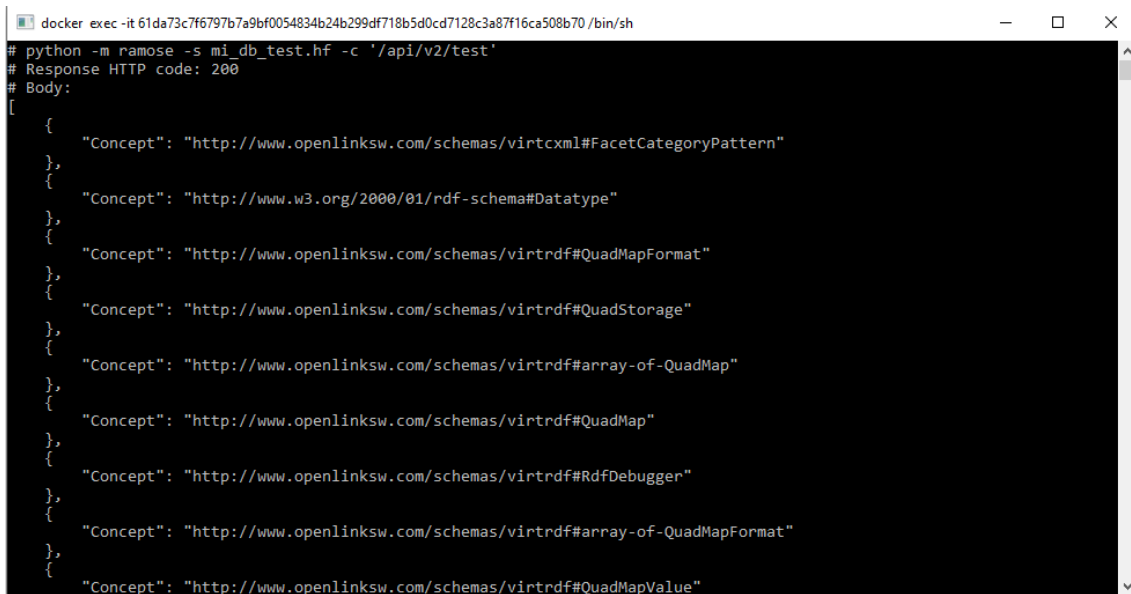
Que realiza una sencilla query:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX schema: <http://schema.org/>
```

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
SELECT distinct ?Concept where {  
  [] a ?Concept  
} LIMIT 100
```

Y tiene como resultado lo siguiente:



```
docker exec -it 61da73c7f6797b7a9bf0054834b24b299df718b5d0cd7128c3a87f16ca508b70 /bin/sh  
# python -m ramose -s mi_db_test.hf -c '/api/v2/test'  
# Response HTTP code: 200  
# Body:  
[  
  {  
    "Concept": "http://www.openlinksw.com/schemas/virtcxml#FacetCategoryPattern"  
  },  
  {  
    "Concept": "http://www.w3.org/2000/01/rdf-schema#Datatype"  
  },  
  {  
    "Concept": "http://www.openlinksw.com/schemas/virttrdf#QuadMapFormat"  
  },  
  {  
    "Concept": "http://www.openlinksw.com/schemas/virttrdf#QuadStorage"  
  },  
  {  
    "Concept": "http://www.openlinksw.com/schemas/virttrdf#array-of-QuadMap"  
  },  
  {  
    "Concept": "http://www.openlinksw.com/schemas/virttrdf#QuadMap"  
  },  
  {  
    "Concept": "http://www.openlinksw.com/schemas/virttrdf#RdfDebugger"  
  },  
  {  
    "Concept": "http://www.openlinksw.com/schemas/virttrdf#array-of-QuadMapFormat"  
  },  
  {  
    "Concept": "http://www.openlinksw.com/schemas/virttrdf#QuadMapValue"
```

La configuración inicial de esta herramienta es bastante más compleja a los ejemplos mostrados, debido a la estructura de sus datos, por lo que su sentencia es más complicada y requiere tener conocimientos sobre los datos previamente a la consulta. Un ejemplo de la configuración inicial sería el siguiente:

```
python -m ramose -s test.hf -c  
'/api/v1/metadata/10.1107/S0567740872003322__10.1007/BF02020444'
```



```
docker exec -it 61da73c7f6797b7a9bf0054834b24b299df718b5d0cd7128c3a87f16ca508b70 /bin/sh
# python -m ramose -s test.hf -c '/api/v1/metadata/10.1107/S0567740872003322_10.1007/BF02020444'
# Response HTTP code: 200
# Body:
[
  {
    "author": "",
    "year": "1972",
    "title": "The crystal structure of tin(II) sulphate",
    "source_title": "Acta crystallographica Section B Structural crystallography and crystal chemistry",
    "volume": "28",
    "issue": "3",
    "page": "864-867",
    "doi": "10.1107/S0567740872003322",
    "reference": "",
    "citation_count": "1",
    "qid": "Q29013687"
  },
  {
    "author": "Erdős, Paul; Erdős, Pál; Hajnal, András",
    "year": "1966",
    "title": "On chromatic number of graphs and set-systems",
    "source_title": "Acta mathematica Academiae Scientiarum Hungaricae",
    "volume": "17",
    "issue": "1-2",
    "page": "61-99",
    "doi": "10.1007/BF02020444",
    "reference": "10.4153/CJM-1959-003-9",
    "citation_count": "2",
    "qid": "Q57259020"
  }
]
```

### 3.5.4 Puntos débiles

El punto débil más destacable en comparación a las demás herramientas es que, Ramose sólo trata con JSON y csv, mientras que el resto de las herramientas analizadas pueden utilizar tripletas, turtle o html. La documentación que posee la herramienta, aunque abundante, no es muy eficaz a la hora de presentar la información de manera más específica. Al presentarse de forma teórica, necesita de la ayuda de ejemplos, pero el ejemplo proporcionado en la configuración inicial no es muy sencillo lo que provoca más dudas que soluciones.

Por cómo está estructurada la herramienta, usando los archivos de configuración con extensión '.hf', donde hay que definir el funcionamiento de la api. Estos dan la sensación de haber trabajado doble, por un lado, se habría creado la API y por otro, para utilizar la herramienta se tendría que volver a definir el diseño de la api en un archivo de configuración. Lo cual da la sensación de haber hecho el mismo trabajo dos veces.

### 3.5.5 Propuestas de Mejora

Cómo principal mejora sería añadir el uso de más tipos de contenido, no solo JSON y csv, de esta manera estaría más equilibrada con las demás herramientas analizadas.

Para que la herramienta sea más fácil de entender, se propone que se añadan sentencias de ejemplo nada más haber explicado el comando de forma teórica, así aumentaría la claridad de la documentación. Además de utilizar un ejemplo más sencillo para explicar la funcionalidad de la herramienta, debido a que el actual es difícil de entender a primera vista, algo que un ejemplo no debería de ser.

## 3.6 WALDER

### 3.6.1 Configuración

Esta herramienta archivos de configuración con la extensión ‘.yaml’, dichos archivos se encuentran dentro de la carpeta “example”. Dichos archivos de configuración definen el funcionamiento de la API, donde se indican todos los métodos que se deseen. Para definir un método es necesario especificar en qué ruta está ubicado, es decir, que es necesario poner en el navegador para acceder a dicho método. También si se necesitan parámetros en la URI deben ser indicados tanto en la parte de la ruta como un apartado indicando que tipo de parámetro es.

A continuación, se define la query en GRAPHQL [] y las dependencias que necesite, indicando el endpoint que se va a utilizar en la parte de JSON-LD, quedando la configuración de la siguiente manera:

```
openapi: 3.0.2
info:
  title: 'Example site'
  version: 0.1.0
x-walder-resources:
  root: ./
  views: views
  pipe-modules: pipeModules
  public: public
x-walder-datasources:
  - http://fragments.dbpedia.org/2016-04/en
paths:
  /music/{musician}:
    get:
      summary: Returns a list of songs of the given musician.
      parameters:
        - in: path
          name: musician
          required: true
          schema:
            type: string
          description: The target musician
      x-walder-query:
        graphql-query: >
        {
          label @single
          artist(label_en: $musician)
        }
        json-ld-context: >
        {
          "@context": {
            "label": "http://www.w3.org/2000/01/rdf-schema#label",
            "label_en": { "@id": "http://www.w3.org/2000/01/rdf-schema#label", "@language": "en" },
            "writer": "http://dbpedia.org/ontology/writer",
            "artist": "http://dbpedia.org/ontology/musicalArtist"
          }
        }
      responses:
        200:
          description: list of songs
          x-walder-input-text/html: songs.handlebars
```

Para definir la configuración es necesario seguir “OpenAPI 3.0 specification” para funcionar. La herramienta además necesita definir las respuestas del servidor OK para indicar un formato a seguir.

Se ha creado un Dockerfile para esta herramienta, para poder ser ejecutada en cualquier máquina. Este Dockerfile tiene como base la imagen de Node.js y se le añaden las dependencias que necesita esta herramienta, en este caso se ejecuta el siguiente comando:

```
yarn global add walder
```

Para montar el Dockerfile utilizar la siguiente línea de comando en el directorio del Dockerfile:

```
docker build -t <nombre> .
```

Donde <nombre> es el nombre que se le asignará a la imagen creada.

Una vez la imagen esté creada, se ejecuta el siguiente comando y se accede al CLI creado en Docker Desktop:

```
docker run -it --rm -p 3000:3000 --name miwalder my_walder
```

(NOTA: en este caso se ha montado la imagen con nombre “my\_walder” y creado el contenedor “miwalder”)

### 3.6.2 Ejecución

Tras abrir la ventana de CLI del contenedor recién creado “miwalder”, solamente es necesario ejecutar el siguiente comando para ejecutar la herramienta:

```
walder -c example/myconfig.yaml
```

La opción “-c” especifica que configuración dentro de la carpeta “example” se procede a ejecutar, en este caso se ejecutará la configuración de la API “myconfig.yaml” que contiene dos métodos. Siendo estos muy similares debido a no tener muchos conocimientos sobre GraphQL.

El puerto por defecto de la aplicación es el 3000, por ello ejecutamos la imagen de esa manera. Si se quiere cambiar el puerto, se debe abrir al ejecutar la imagen y elegir el puerto al ejecutar “walder” de esta manera:

```
walder -p <puerto> -c example/myconfig.yaml
```

Una sentencia válida sería la siguiente:

```
walder -p 8080 -c example/myconfig.yaml
```

### 3.6.3 Ejemplos

Una vez ejecutado el comando mencionado anteriormente, se muestra el estado de los métodos definidos en la configuración ejecutada, siendo este caso de esta forma:

```
# walder -c example/myconfig.yaml
info 2021-06-02T09:27:20.350Z : Starting static server...
info 2021-06-02T09:27:20.352Z : Static server running.
info 2021-06-02T09:27:20.353Z : Parsing route /Jackson
info 2021-06-02T09:27:20.353Z :      - Parsing method get
info 2021-06-02T09:27:20.357Z : Parsing route /Gaga
info 2021-06-02T09:27:20.357Z :      - Parsing method get
info 2021-06-02T09:27:20.360Z : Starting server...
info 2021-06-02T09:27:20.361Z : Started listening on port: 3000.
```

Una vez el servicio esté arrancado, para esta configuración con dos métodos se encuentran en la siguiente dirección:

<http://localhost:3000/Jackson>

<http://localhost:3000/Gaga>

Estos métodos realizan la siguiente sentencia:

```
paths:
  /Jackson:
    get:
      summary: Returns a list of songs written by Michael Jackson.

      x-walder-query:
        graphql-query: >
          { label @single
            writer(label_en: "Michael Jackson")
            artist { label }
          }
        json-ld-context: >
          {
            "@context": {
              "label": "http://www.w3.org/2000/01/rdf-schema#label",
              "label_en": { "@id": "http://www.w3.org/2000/01/rdf-schema#label", "@language": "en" },
              "writer": "http://dbpedia.org/ontology/writer",
              "artist": "http://dbpedia.org/ontology/musicalArtist"
            }
          }
      responses:
        200:
          description: list of songs
          x-walder-input-text/html: songs.handlebars
```

Estos métodos devuelven una lista de canciones escritas por Michael Jackson y Lady Gaga respectivamente. Cuyos resultados son los siguientes:

## Songs

(I Like) The Way You Love Me

---

A Place with No Name

---

All in Your Name

---

Another Part of Me

---

Bad (Michael Jackson song)

---

Beat It

---

Behind the Mask (Michael Jackson song)

---

Billie Jean

---

Black or White

---

Blood on the Dance Floor (song)

---

Can You Feel It (The Jacksons song)

---

Centipede (song)

---

Dirty Diana

---

Do Thangz

---

Don't Be Messin' 'Round

---

Don't Stop 'Til You Get Enough

---

Don't Stop Til You Get Enough

---

Earth Song

---

Eat It

---

Eaten Alive (song)

---

Resultado de las canciones escritas por Michael Jackson

## Songs

Alejandro (song)
Applause (Lady Gaga song)
Bitch, Don't Kill My Vibe
Do What U Want
Eh, Eh (Nothing Else I Can Say)
Fever (Adam Lambert song)
G.U.Y.
Judas (Lady Gaga song)
LoveGame
Make Her Say
Marry the Night
Paparazzi (Lady Gaga song)
Perform This Way
Poker Face (Lady Gaga song)
The Edge of Glory
Til It Happens to You
You and I (Lady Gaga song)

Y este el resultado de las canciones escritas por Lady Gaga

### 3.6.4 Puntos débiles

El punto más débil de esta herramienta es que está limitada a ejecutarse sobre un software específico, dado que las instrucciones de instalación no funcionan en un sistema operativo Windows, necesitando así Ubuntu o Linux.

Además, aunque la documentación es extensa y de buena calidad sobre cómo realizar la configuración de la herramienta, las queries se deben realizar en un lenguaje distinto al resto de herramientas analizadas. En este caso usando GRAPHQL en vez de SPARQL lo que supone una debilidad ya que el modo de uso es muy diferente en comparación a las demás herramientas, lo que equivale a aprender otro lenguaje.

También es la única de las herramientas analizadas en la que es necesario especificar las respuestas del servidor, es decir, que devuelve el servido en caso de error y de OK.

### 3.6.5 Propuestas de Mejora

Debido a que esta herramienta es la única de las analizadas que no se ha podido instalar en un entorno Windows por la sentencia de instalación necesaria. La propuesta principal de mejora para esta herramienta sería la creación, por parte de las personas a cargo del proyecto, de un Dockerfile y asegurar su mantenimiento en caso de que se necesite.

Por otra parte, al usar GraphQL para las queries en la configuración, se podrían añadir comparativas de una sentencia en GraphQL con SPARQL para tener una mejor visualización de qué operación están realizando dichas queries y así tener un mayor entendimiento a la hora de crear una configuración.

## 4 Resultados y conclusiones

Por lo general se han podido instalar todas las herramientas propuestas para este trabajo y probarlas casi en su totalidad. Habiendo trabajado con estas herramientas durante el trabajo, se procede a realizar un comentario sobre la dificultad de cada una de ellas y se indicará la última vez que ha recibido soporte la herramienta.

-ELDA: Es una herramienta que gracias a la documentación por pasos que posee es muy sencilla de instalar, sin embargo, al configurar es otra cosa. La documentación es extensa pero no de una gran calidad, debido la instalación de la herramienta contiene muchas carpetas y subcarpetas muy similares entre sí, pero distintas. Entonces la documentación comienza a explicar archivos y cuál es su funcionalidad, pero no indica el directorio en el que se encuentran dichos archivos, por lo que es un poco caótico recorrerse todas las subcarpetas hasta encontrar el archivo que se describen en la documentación.

Además, el ejemplo proporcionado en la documentación es extenso y algo complicado de entender, consiste en juegos de mesa con sus características, el creador del juego con sus características y el distribuidor también con sus características. Pero sobre todo está muy ligado a ese tipo de datos que se utilizan en el ejemplo y la relación que tienen entre ellos, lo que provoca el intentar cambiar esos datos por otros de un endpoint distinto en una tarea bastante complicada.

En resumen, es una herramienta de dificultad alta-media, no por la complejidad en sí, si no que la documentación no ofrece de manera clara y sencilla lo que se buscaba en este objetivo que era cambiar el endpoint. Finalizando la herramienta no ha recibido una actualización por dos años.

-LODI/RDF-Explorer: LODI siendo de las primeras herramientas con las que se comenzó este trabajo y no teniendo mucha idea de cómo hacer las cosas, ha sido la que peor impresión me ha dado. Esto se debe a los problemas mencionados en el apartado tres, que la herramienta posea un Dockerfile, pero no funcione correctamente es una muy mala impresión.

Por ello se pasó a analizar RDF-Explorer, está sí que tiene un Dockerfile funcional, pero carece de una documentación de buena calidad. Dado que lo poco que tiene no está indicado de manera específica, puesto que para cambiar un endpoint indica que crear una entrada en “config”. No indica en cuál de los cuatro archivos dentro de “config” hay que introducirlo, si es dentro de uno de ellos en que parte exactamente o si es crear un archivo completamente nuevo.

En cuanto al uso de la herramienta no es extremadamente difícil, por lo que un usuario medio se puede manejar bastante bien por la herramienta. Por último, LODI no ha recibido una actualización por dos años y RDF-Explorer sin embargo a recibido una actualización en el último año.

-GRLC: Esta ha sido la herramienta que más sencilla me ha resultado tanto de instalar como de configurar nuevas queries con distintos endpoints. En su gran medida por lo comentado en el apartado tres, la gran documentación de buena calidad que posee, además de un tutorial de como iniciar la herramienta paso a paso que está muy bien detallada.

Para finalizar, destacar la facilidad de uso de la herramienta y la cantidad de ejemplos sencillos y fáciles de entender al momento. GRLC ha recibido varias actualizaciones en el transcurso de este año.

-Apache Marmotta: Lo más importante volver a indicar que está herramienta ya está en desuso y no volverá a tener soporte ya que ha sido introducida en Apache Attic donde se encuentra las herramientas que ya no tiene soporte de ningún tipo. La primera impresión de la herramienta es rara dado que es completamente distinta a las demás, tanto en su modo de uso como en la interfaz que presenta. Y, sobre todo, como ya se ha comentado, la herramienta no genera ninguna API para acceder a los datos, solo se navega por ellos. La documentación proporcionada es abundante, pero a su vez algo liosa, por lo que no acaba ayudando tanto como podría.

-RAMOSE: El uso de la herramienta difiere al resto sobre todo por su archivo de configuración tiene una estructura distinta a las demás herramientas hasta ese momento probadas. Esto no es un gran problema debido a que tiene una documentación de buena calidad en la que explica cada parte del archivo de configuración. Sin embargo, el ejemplo propuesto no es tan sencillo de entender en su mayor parte a la complejidad de la query que se muestra en el ejemplo.

Tras entender un poco mejor el ejemplo, es bastante sencillo crear un nuevo archivo de configuración con la query que se quiera realizar y a que SPARQL endpoint se realiza también es muy sencillo de modificar.

Por lo que en general es una herramienta que cuesta un poco entender de primeras, pero posteriormente es sencillo realizar las pruebas que se quiera. Como GRLC, RAMOSE ha recibido varias actualizaciones durante este año.

-Walder: Es la única herramienta que se ha tenido que crear su Dockerfile previamente antes de probarla en local, puesto que su sentencia de instalación no es compatible con un sistema operativo Windows. Una vez creado el Dockerfile la herramienta es bastante sencilla de usar y está bastante bien documentada, lo más difícil de entender fue el lenguaje que utiliza para sus queries, GRAPHQL, pero en general, se ha tenido una buena sensación con la herramienta. Al igual que RAMOSE y GRLC, Walder también a recibido varias actualizaciones durante el año.

Como conclusión del trabajo realizado, me ha parecido bastante interesante el tener que trabajar con herramientas que no se han ni visto en la carrera como Docker, ya que aumenta mis conocimientos de herramientas que se utilizan día a día en diferentes empresas. Por lo que tener una primera toma de contacto con este tipo de herramientas se agradece.

En cuanto al trabajo realizado sobre las herramientas analizadas, me ha resultado curioso ver como cada desarrollador tiene sus formas de hacer las cosas, hasta tal punto que parecen distintos idiomas, esto sobre todo se refleja en la estructura y calidad de la documentación de cada una de las herramientas. Mientras que algunas sólo indican pequeñas pautas de cómo ejecutar sus herramientas, otras realizan una explicación más exhaustiva del funcionamiento de su herramienta o incluso la realización de un tutorial paso a paso con imágenes de como instalar y ejecutar la herramienta.



En general me ha parecido un trabajo entretenido en el que he aprendido el funcionamiento de nuevas tecnologías y como se ha de buscar información cuando las cosas no funcionan como deberían.

## 5 Análisis de Impacto

Con la realización de este trabajo, no se espera provocar un gran cambio a nivel medioambiental, puesto que este no ha sido uno de sus objetivos principales.

Sin embargo, sí que se podría argumentar que tiene cierta relación con el punto 7 de los Objetivos de Desarrollo Sostenible, siendo esta energía asequible y no contaminante. Dado que el esfuerzo realizado en este trabajo sobre la investigación y la forma de configurar las herramientas ahorrará tiempo a todos los desarrolladores que utilicen las herramientas analizadas en este proyecto. Por lo que no tendrán que pasar el tiempo investigando y sí trabajando con dichas herramientas, por lo que ese tiempo ahorrado es tiempo que el ordenador no ha estado consumiendo energía, relacionándose con el punto 7.3 de mejorar la eficiencia energética.

Este ahorro dependerá exclusivamente de los desarrolladores que utilicen este trabajo como base y no pasen el tiempo investigando las herramientas, por ello no se puede estimar la cuantía del ahorro sin realizar un estudio más detallado.

## 6 Bibliografía

- [1] Paola Espinoza-Arias, Daniel Garijo and Oscar Corcho *Crossing the Chasm Between Ontology Engineering and Application Development: A Survey Paper*, Madrid, 2021
- [2] McGuinness, D.L., Van Harmelen, F., et al., 2004. *OWL Web Ontology Language Overview*. W3C recommendation 10, 2004.
- [3] Seaborne, A., Harris, S., 2013. SPARQL 1.1 Query Language. W3C Recommendation. W3C. <https://www.w3.org/TR/2013/REC-sparql11-query-20130321/>
- [4] Bray, T., et al., 2014. The JavaScript Object Notation (JSON) Data Interchange Format.
- [5] Salvadori, I., Siqueira, F., 2015. A Maturity Model for Semantic RESTful Web APIs, in: 2015 IEEE International Conference on Web Services, IEEE. pp. 703–710.
- [6] Solomon Hykes, software libre y de código abierto, Docker, Inc. , 13/04/2013
- [7] Fernández-Sellers, M., 2015. Linked Open Data Inspector. URL: <https://github.com/marfersel/LODI/>. accessed: 2020-03-12.
- [8] Maximiliano Osorio, RDF-Explorer. URL: [https://github.com/KnowledgeCaptureAndDiscovery/rdf\\_explorer](https://github.com/KnowledgeCaptureAndDiscovery/rdf_explorer) accessed : 2021-03-04
- [9] Epimorphics, 2011. Elda: The linked-data API in Java. URL: <http://epimorphics.github.io/elda/index.html>. accessed: 2020-03-09.
- [10] Foundation, T.A.S., 2013. Apache Marmotta. URL: <http://marmotta.apache.org/>. accessed: 2020-07-10.
- [11] GRLC , Albert Menroyo, URL: <https://github.com/CLARIAH/grlc>
- [12] Daquino, M., Heibi, I., Peroni, S., Shotton, D., 2020. Creating Rest ful APIs over SPARQL endpoints with RAMOSE. arXiv preprint arXiv:2007.16079 . URL: <https://github.com/opencitations/ramose>
- [13] Heyvaert, P., De Meester, B., Pandit, H., Verborgh, R., 2020. Walder. URL: <https://github.com/KnowledgeOnWebScale/walder>. accessed: 2021-5-15.
- [14] James Gosling, 1995, Java, URL: <https://www.java.com/es/>
- [15] Jason van Zyl, 2003, Apache Maven, URL: <https://maven.apache.org/>

