

Two large, stylized blue handprints are positioned on either side of the title text, with the fingers spread out.

SPARQL Micro-Services hands-on

Franck MICHEL

UNIVERSITÉ
CÔTE D'AZUR



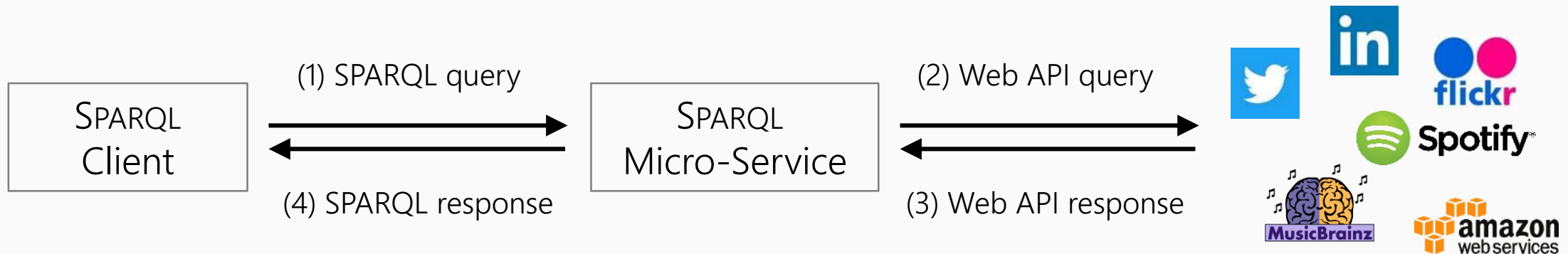
Inria



Main concepts

The SPARQL Micro-Service Architecture

Lightweight method to **query a Web API with SPARQL**,
and assign dereferenceable URIs to Web API resources



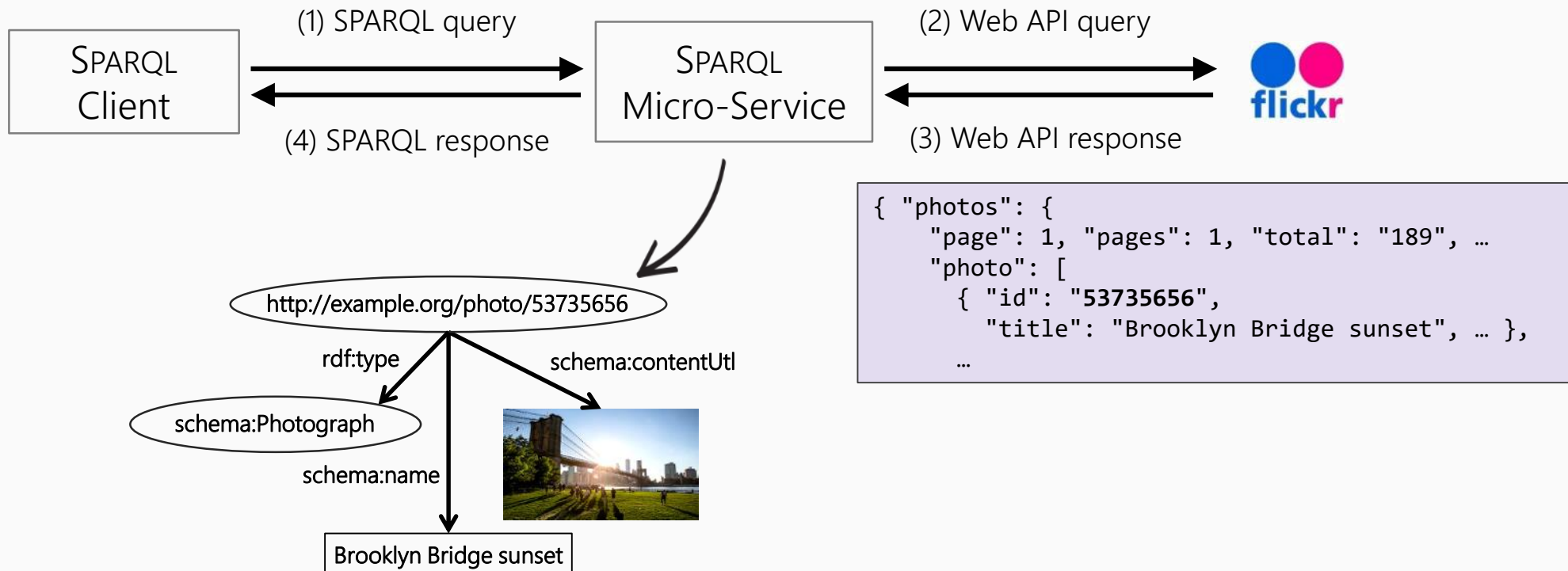
<https://github.com/frmichel/sparql-micro-service>

Details - Arguments passed as HTTP parameters

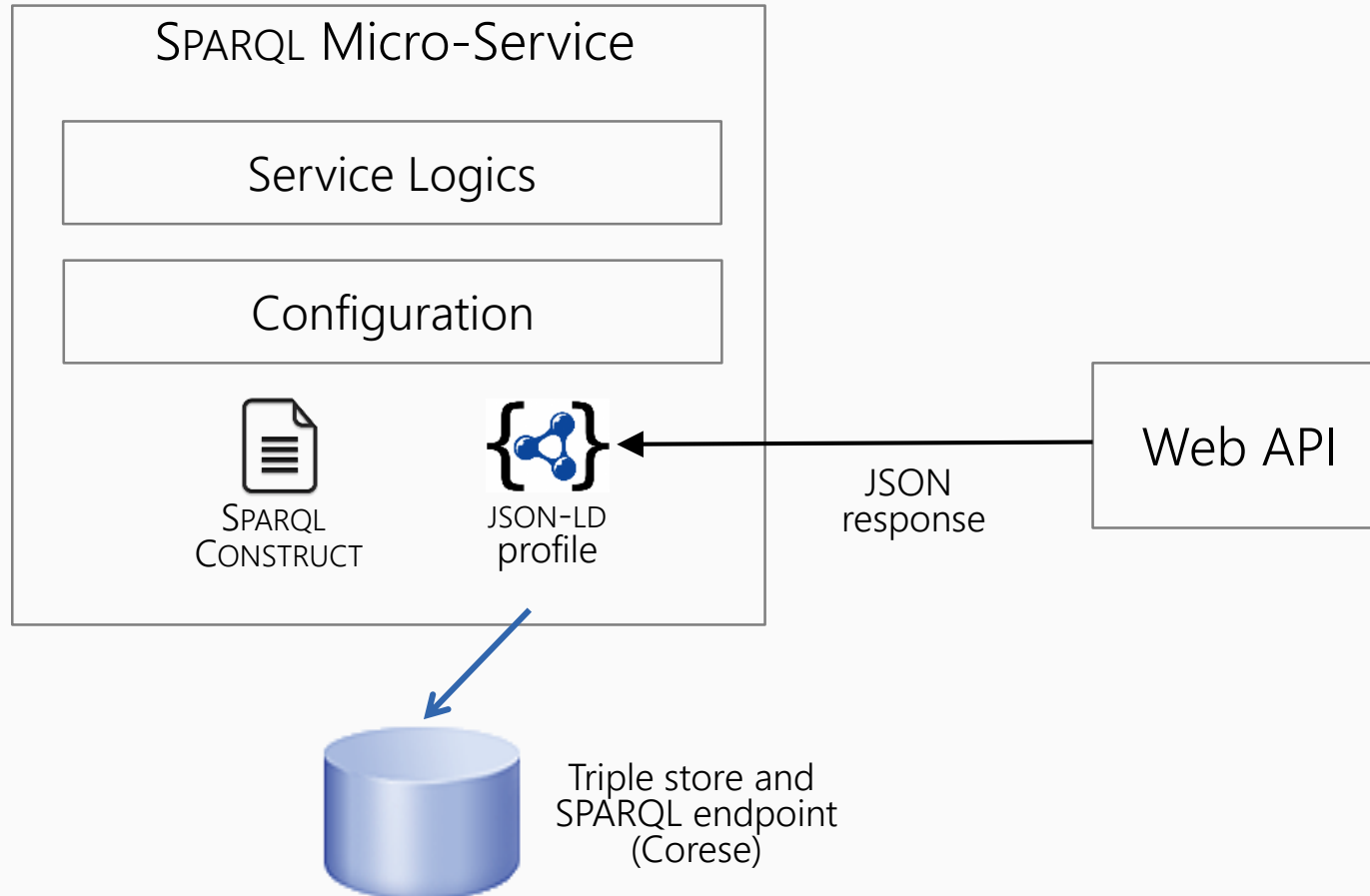
```
SELECT * WHERE {  
  ?photo a schema:Photograph;  
  schema:name ?title;  
  schema:contentUrl ?img.  
}
```

Endpoint: <http://example.org/flickr/getPhotosByTag?tag=bridge>

https://api.flickr.com/services/rest/?method=flickr.photos.search&format=json&per_page=100&tags=bridge&...



Translating the Web API response to RDF



First translation with a JSON-LD context

```
{  
  "id": "53735656",  
  "title": "Brooklyn Bridge sunset",  
}
```

API response



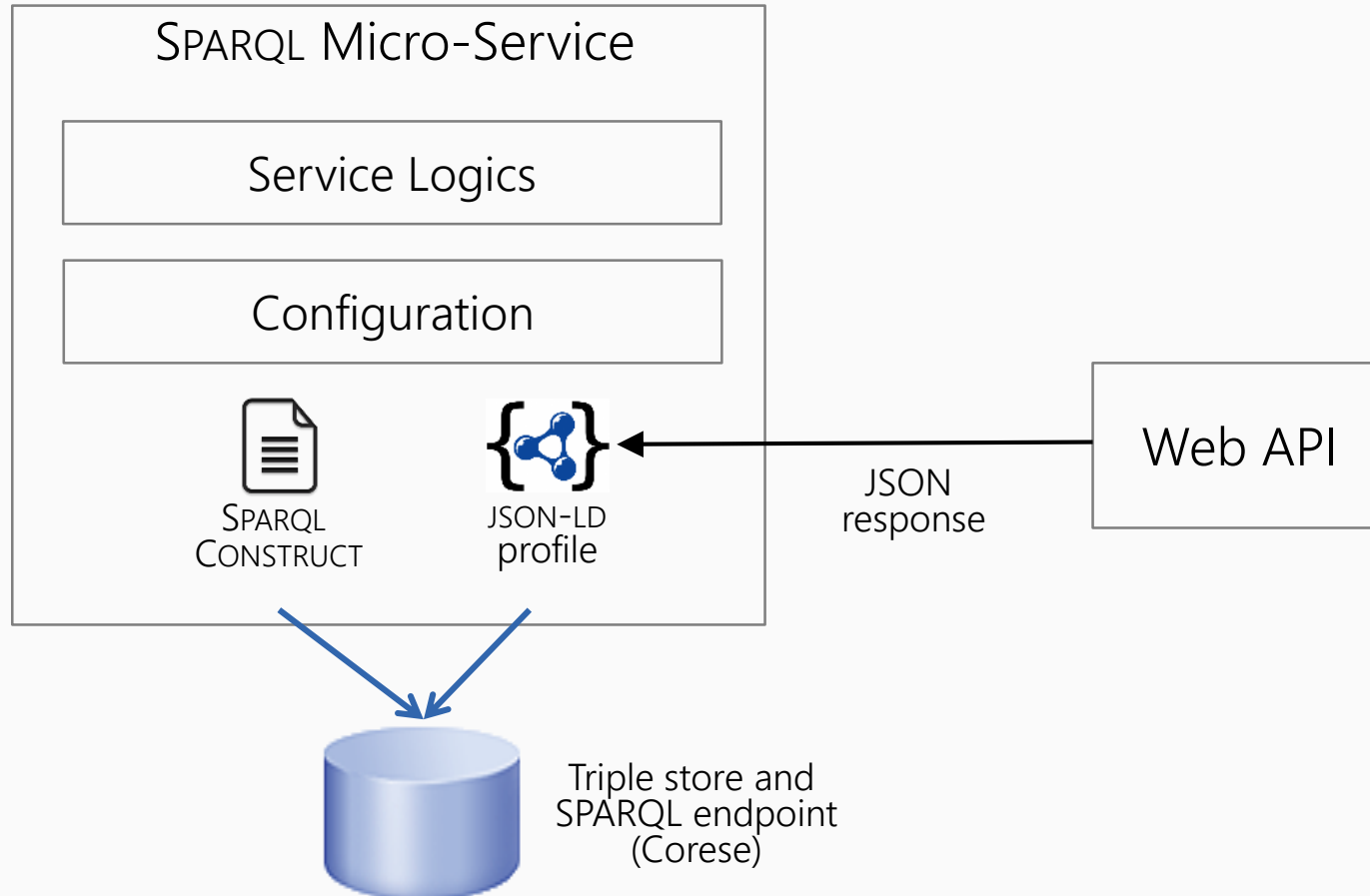
```
{  
  "@context": {  
    "@vocab": "http://ns.inria.fr/sparql-micro-service/api#"  
  }  
}
```

profile.jsonld

```
@prefix api: <http://ns.inria.fr/sparql-micro-service/api#>  
  
[] api:id      "53735656";  
   api:title   "Brooklyn Bridge sunset";
```



Translating the Web API response to RDF



Advanced mapping with SPARQL (optional)

```
@prefix api: <http://ns.inria.fr/sparql-micro-service/api#>
```



```
[ ] api:id      "53735656";  
    api:title   "Brooklyn Bridge sunset";
```

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

construct.sparql

```
CONSTRUCT {
```

```
  ?photoUri
```

```
    a          schema:Photograph;
```

```
    schema:name ?title;
```

```
}
```

```
WHERE {
```

```
  ?result
```

```
    api:id      ?photoId;
```

```
    api:title   ?title;
```

```
  BIND(IRI(concat("http://example.org/ld/photo/", ?photoId)) AS ?photoUri)
```

```
}
```

```
@prefix schema: <http://schema.org/>
```




```
<http://example.org/ld/photo/53735656>
```

```
  a          schema:Photograph;
```

```
  schema:name "Brooklyn Bridge sunset".
```


Quick-start guide

Main approach to create a SPARQL μ -service

- 
- Read the API documentation:**
find out the API service that does what you want to do
 - Create a basic SPARQL micro-service:**
a JSON-LD profile translates the API response into "raw" RDF (api namespace)
 - Decide of a mapping to an RDF vocabulary:**
figure out an appropriate vocabulary for your use case
 - Write the mapping in a CONSTRUCT query**
and test the service.

My first SPARQL micro-service

Goal: find music albums on Deezer, that match a keyword



- Find out the right API query
- Create a basic SPARQL micro-service
- Decide of a mapping to an RDF vocabulary
- Write the mapping in a CONSTRUCT query

Check the Web API documentation

<https://developers.deezer.com/api/search#connections>

Choose a service to be fulfilled by the SPARQL micro-service

Query music albums by keyword (search > Search Methods)

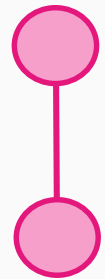
Find out the right query to do this

<https://api.deezer.com/search/album?q=eminem>

My first SPARQL micro-service



Documentation:
<https://s.42l.fr/smsdoc>



Find out the right
API query



Create a basic
SPARQL micro-service



Decide of a mapping to
an RDF vocabulary



Write the mapping in a
CONSTRUCT query

1. CD to the directory of deployed services.

2. Create directory `deezer/findAlbums`

3. In `deezer/findAlbums`, create file `config.ini` with 2 properties:

```
api_query = "https://api.deezer.com/search/album?q={keyword}"  
custom_parameter[] = keyword
```

Create file `profile.jsonld`

```
{ "@context": {  
  "@vocab": "http://ns.inria.fr/sparql-micro-service/api#"  
}}
```

Query the SPARQL micro-service (using Yasgui):

Endpoint URL:

```
http://localhost/service/deezer/findAlbums?keyword=eminem
```

Query:

```
select * where { ?s ?p ?o. }
```

My first SPARQL micro-service



Find out the right
API query



Create a basic
SPARQL micro-service



Decide of a mapping to
an RDF vocabulary



Write the mapping in a
CONSTRUCT query

Find appropriate vocabularies

- Schema.org <https://schema.org/docs/full.html>
- Wikidata <https://wikidata.org>
- LOV (Linked Open Vocabularies)
<https://lov.linkeddata.es/dataset/lov/>
- Specialized ontology portals...

Schema.org

[Thing](#) > [CreativeWork](#) > [MusicPlaylist](#) > [MusicAlbum](#)

[Thing](#) > [Organization](#) > [PerformingGroup](#) > [MusicGroup](#)

[Thing](#) > [CreativeWork](#) > [MusicRecording](#)

...

My first SPARQL micro-service

- Find out the right API query
- Create a basic SPARQL micro-service
- Decide of a mapping to an RDF vocabulary
- Write the mapping in a CONSTRUCT query

```
PREFIX schema: <http://schema.org/>
CONSTRUCT {
  []
    a                schema:MusicAlbum;
    schema:name       ?albumTitle;
    schema:image       ?imageUri;
    schema:byArtist    ?artistName
}
WHERE {
  ?album
    api:title  ?albumTitle;
    api:cover  ?image;
    api:artist [ api:name ?artistName ].

    bind(iri(?image) as ?imageUri)
}
```

Query the SPARQL micro-service with Yasgui

The screenshot shows the Yasgui SPARQL client interface. At the top, there are tabs for 'Query' and 'SPμS'. The 'Query' tab is active, displaying a URL bar with the endpoint `http://localhost/service/deezer/findAlbums?keyword=orelsan`. Below the URL bar, a SPARQL query is entered in a text area:

```
1 PREFIX schema: <http://schema.org/>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4 PREFIX api: <http://ns.inria.fr/sparql-micro-service/api#>
5 SELECT * WHERE {
6   ?s ?p ?o.
7 }
```

On the right side of the query editor, there are icons for sharing and executing the query. Below the query editor, the results are displayed in a table view. The table has three columns: 's', 'p', and 'o'. The results are filtered to show the first 4 entries out of 100 total entries. The table shows the following data:

| | s | p | o |
|----|-------|-----------------|---|
| 1 | b3788 | schema:byArtist | "Orelsan"^^<http://www.w3.org/2001/XMLSchema#string> |
| 26 | b3788 | schema:image | <https://api.deezer.com/album/270762122/image> |
| 51 | b3788 | schema:name | "Civilisation"^^<http://www.w3.org/2001/XMLSchema#string> |
| 76 | b3788 | rdf:type | schema:MusicAlbum |

At the bottom of the interface, there is a pagination bar showing 'Showing 1 to 4 of 4 entries (filtered from 100 total entries)' and navigation controls.

Query the SPARQL micro-service with Yasgui

The image displays two screenshots of the Yasgui SPARQL client interface, showing queries and their results.

Left Screenshot:

- URL: `http://localhost/service/deezer/findAlbums?keyword=orelsan`
- Query:

```
1 PREFIX schema: <http://schema.org/>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4 PREFIX api: <http://ns.inria.fr/sparql-micro-service/api#>
5 SELECT * WHERE {
6   ?s ?p ?o.
7 }
```
- Results: 100 results in 0.129 seconds. The table shows columns **s**, **p**, and **o**. Visible entries include:

| | s | p | o |
|----|-------|-----------------|---|
| 1 | b3788 | schema:byArtist | "Orelsan"^^<http://www.w3.org/2001/XMLSchema#string> |
| 26 | b3788 | schema:image | <https://api.deezer.com/album/b3788/image> |
| 51 | b3788 | schema:name | "Civilisation"^^<http://www.w3.org/2001/XMLSchema#string> |
| 76 | b3788 | rdf:type | schema:MusicAlbum |

Right Screenshot:

- URL: `http://localhost/service/deezer/findAlbums?keyword=orelsan`
- Query:

```
1 PREFIX schema: <http://schema.org/>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4 PREFIX api: <http://ns.inria.fr/sparql-micro-service/api#>
5 SELECT * WHERE {
6   ?s a schema:MusicAlbum;
7     schema:name ?name.
8 }
```
- Results: 25 results in 0.132 seconds. The table shows columns **s** and **name**. Visible entries include:

| | s | name |
|---|-------|--|
| 1 | b3687 | "N'importe comment (The Remixes)"^^<http://www.w3.org/2001/XMLSchema#string> |
| 2 | b3688 | "Potentiel (feat. Orelsan)"^^<http://www.w3.org/2001/XMLSchema#string> |
| 3 | b3689 | "Enfant lune"^^<http://www.w3.org/2001/XMLSchema#string> |
| 4 | b3690 | "Millions"^^<http://www.w3.org/2001/XMLSchema#string> |
| 5 | b3691 | "Tous les jours dimanche"^^<http://www.w3.org/2001/XMLSchema#string> |
| 6 | b3692 | "Double Vie - Single"^^<http://www.w3.org/2001/XMLSchema#string> |

Query the SPARQL micro-service with Yasgui

The screenshot shows the Yasgui SPARQL client interface. At the top, there are tabs for 'Query' and 'SμS', with a '+' icon to add more. Below the tabs is a text input field containing the URL: `http://localhost/service/deezer/findAlbums?keyword=orelsan`. To the right of the input field is a share icon and a play button.

The query editor shows the following SPARQL query:

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX api: <http://ns.inria.fr/sparql-micro-service/api#>
4 CONSTRUCT WHERE {
5   ?s ?p ?o.
6 }
```

Below the query editor, there are tabs for 'Table' and 'Response', with a status bar indicating '100 results in 0.122 seconds'. To the right of the tabs are a download icon and a help icon.

The response is displayed in the 'Response' tab, showing two results in N-Triples format:

```
1 @prefix ns1: <http://schema.org/> .
2 @prefix api: <http://ns.inria.fr/sparql-micro-service/api#> .
3 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
4
5
6 _:b3906 ns1:byArtist "Orelsan" ;
7 ns1:image <https://api.deezer.com/album/1261738/image> ;
8 ns1:name "Suicide Social - Single" ;
9 rdf:type ns1:MusicAlbum .
10
11 _:b3905 ns1:byArtist "Vald" ;
12 ns1:image <https://api.deezer.com/album/291023182/image> ;
13 ns1:name "V" ;
14 rdf:type ns1:MusicAlbum .
15
```

Docker environment

| | |
|----------------------------|---|
| Instructions | https://github.com/frmichel/sparql-micro-service/tree/master/deployment/docker |
| SPARQL μ -services URL | <code>http://localhost/service/<api>/<service>?...</code> |
| YASGUI | file:///home/user/yasgui.html |