



Initiative for digital transformation in the Metadata and
Reference Data Sector of the Publications Office of the
European Union

Installation guide for the RDF validation service

Disclaimer

The views expressed in this report are purely those of the Author(s) and may not, in any circumstances, be interpreted as stating an official position of the European Union. The European Union does not guarantee the accuracy of the information included in this study, nor does it accept any responsibility for any use thereof. Reference herein to any specific products, specifications, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favouring by the European Union.

This report was prepared for the Publications Office of the European Union by Infeurope.

Document metadata

Reference	WP-RD-8: Installation guide for the RDF validation service
Corporate Author	Publications Office of the European Union
Author	Eugeniu Costetchi
Reviewers	Denis Dechandon and Willem Van Gemert
Contractor	Infeurope S.A.
Framework contract	10688/37490
Work package	WP-RD-8
Delivery date	10 January 2022
Suggested readers	technical staff, system administrators, enterprise architects, software developers

Abstract

This document provides technical guidance on how to install and configure the suite of micro-services and applications necessary for the asset metadata lifecycle process at the Standardisation Unit at the Publications Office of the European Union.

Contents

1	Introduction	5
2	Scope	5
3	Target audience	5
4	Technology background	6
5	Requirements	6
6	Installation	6
7	Configuration	8
	7.1 RDF validator	8
	7.2 Celery worker	8
	7.3 Configure and read logs	9
8	Add a new application profile	10
9	Custom templates	11
	9.1 HTML template variant	11
	9.2 JSON template variant	12
10	API documentation	13
	10.1 Get application profiles	13
	10.2 Get all validation reports	14
	10.3 Get validation report	15
	10.4 Delete validation report	16
	10.5 Delete all validation reports	17
	10.6 Validate file	17
	10.7 Validate file with Application Profile	18
	10.8 Validate SPARQL Endpoint	19
	10.9 Validate SPARQL Endpoint with Application Profile	20
	10.10 List active tasks	21
	10.11 Get task status	22
	10.12 Stop task execution	22

1 Introduction

The Standardisation Unit (SU) at the Publications Office of the European Union (OP) is engaged in a digital transformation process oriented towards semantic technologies. In [1] is described a working definition of the architectural stance and design decisions that are to be adopted for the asset publication life-cycle process. The report describes the baseline (current) solution and the (new) target solution for the asset publication workflow that is part of the life-cycle process.

The software components building up the target publication workflow solution have been packaged as into a suite of interconnected services.

This document describes the installation and configuration procedures along with stating the scope, and target audience.

2 Scope

This document aims at covering the installation and configuration instructions for the suite of the following software services:

1. RDF validator
2. Validator celery worker

3 Target audience

The target audience for this document comprises the following groups and stakeholders:

- Technical staff in charge of operating workflow components
- System administrators
- Enterprise architects and data governance specialists
- Documentalists involved in the reference data life-cycle
- Developers in charge of workflow and component implementation
- Third parties using the SU services and data

4 Technology background

Infrastructure and deployment configuration rely on services deployed on a CentOS system.

5 Requirements

There is a range of ports that must be available on the host machine as they will be bound to by different services. Although the system administrator may choose to change them by changing the values in of specific environment variables. The inventory of pre-configured ports is provided in Table 1.

Service name	HTTP	HTTP
	port UI	port API
RDF validator	8010	4010
redis		6379

Table 1: Port usage inventory

The minimal hardware requirements are as follows

1. CPU: 3.2 Ghz quad core
2. RAM: 16GB
3. SDD system: 32GB
4. SDD data: 128GB

6 Installation

In order for the services to function properly a CentOS system with python version 3.6 and redis service should be setup and running with the appropriate ports and addresses configured in the environment variable file.

Copy the rdf validator zip on the system you intend to run it and unzip it.

Contents

Then change directory into the *project* folder. Makefile commands to start and stop services will be available.

To start the services using Makefile

```
make install-python-dependencies
make setup-rdfunit
make run-api
make run-ui
```

To stop the services using Makefile

```
make stop-gunicorn
```

To start services without Makefile commands

```
set -o allexport; source bash/.env; set +o allexport

python3 -m venv env
source env/bin/activate
pip install -r requirements/prod.txt
```

then start the services

```
set -o allexport; source bash/.env; set +o allexport

source env/bin/activate

celery -A validator.adapters.celery.celery_worker worker --loglevel
    ${RDF_VALIDATOR_LOG_LEVEL} --logfile ${
    RDF_VALIDATOR_CELERY_LOGS} --detach
gunicorn --timeout ${RDF_VALIDATOR_GUNICORN_TIMEOUT} --workers ${
    RDF_VALIDATOR_GUNICORN_API_WORKERS} --bind 0.0.0.0:${
    RDF_VALIDATOR_API_PORT} --reload validator.entrypoints.api.run:
    app --log-file ${RDF_VALIDATOR_API_LOGS} --log-level ${
    RDF_VALIDATOR_LOG_LEVEL} --daemon
gunicorn --timeout ${RDF_VALIDATOR_GUNICORN_TIMEOUT} --workers ${
    RDF_VALIDATOR_GUNICORN_UI_WORKERS} --bind 0.0.0.0:${
    RDF_VALIDATOR_UI_PORT} --reload validator.entrypoints.ui.run:app
    --log-file ${RDF_VALIDATOR_UI_LOGS} --log-level ${
    RDF_VALIDATOR_LOG_LEVEL} --daemon
```

To stop the services run

```
source env/bin/activate

celery -9 -f celery
pkill -f gunicorn
```

7 Configuration

At deployment and at runtime, the service configurations are provided through OS environment variables available in the `.env` file. The role of the `.env` file is to enable the system administrators to easily change default configurations as necessary in the context of their environment.

The suite of micro-services is built, started and shut down via Makefile commands.

In order to avoid hard coding parameters, they are defined externally in the `.env`. Having them in a single file makes sense and it is more pragmatic, as you can see and manage all parameters in one place, add the file to the version control system (the contents of the file will evolve and be in sync with the actual code) and have different files for different environments.

The following sections describe the configuration options available for each of the services.

7.1 RDF validator

The RDF validator is an online platform for validating RDF data with SHACL shape definitions. It exposes an API and an UI. The RDF validator API is the core service providing the RDF validation functionality. The URL and port are described below, as well as the request timeout:

Description	Value	Associated variable
Service URL	http://localhost	RDF_VALIDATOR_API_LOCATION
Service API port	4010	RDF_VALIDATOR_API_PORT
Is in debug mode	False	RDF_VALIDATOR_DEBUG
Service UI port	8030	RDF_VALIDATOR_UI_PORT
Web server worker process timeout	1200	RDF_VALIDATOR_GUNICORN_TIMEOUT

Table 2: RDF validator configurations

7.2 Celery worker

Celery is a simple, flexible, and reliable distributed system to process vast amounts of messages, while providing operations with the tools required to maintain such a

system. It's a task queue with focus on real-time processing.

In the RDF validator project it serves the purpose of enabling multiprocessing of validation report generation.

The RDF validator application uses the following Celery environment variables

Description	Value	Associated variable
Redis location	redis://localhost	RDF_VALIDATOR_REDIS_LOCATION
Redis port	6379	RDF_VALIDATOR_REDIS_PORT

Table 3: Celery environment configurations

More about the implementation of multiprocessing can be found in the *adapters/celery.py*. A fragment of how celery is used and the asynchronous file validation is presented below:

```
celery_worker = Celery('rdf-validator-tasks',
    broker=config.RDF_VALIDATOR_REDIS_SERVICE,
    backend=config.RDF_VALIDATOR_REDIS_SERVICE)
celery_worker.conf.update(result_extended=True)

CELERY_VALIDATE_FILE = 'validate_file'

@celery_worker.task(name=CELERY_VALIDATE_FILE, bind=True)
def async_validate_file(self, uid: str, data_file: str,
    shacl_shapes: list, db_cleanup_location: str,
    application_profile: str = ''):
    ...
```

7.3 Configure and read logs

Every service provided by the RDF validator has it's own log history and is configurable through the aforementioned *.env* file. The current configuration accepts a relative path to where the logs to be written *logs/api.log*, for example.

API log example

```
[2022-01-02 15:34:35 +0000] [8] [INFO] Starting gunicorn 20.1.0
[2022-01-02 15:34:35 +0000] [8] [DEBUG] Arbiter booted
```

```
[2022-01-02 15:34:35 +0000] [8] [INFO] Listening at: http
://0.0.0.0:4010 (8)
[2022-01-02 15:34:35 +0000] [8] [INFO] Using worker: sync
[2022-01-02 15:34:35 +0000] [11] [INFO] Booting worker with pid: 11
[2022-01-02 15:34:35 +0000] [12] [INFO] Booting worker with pid: 12
[2022-01-02 15:34:35 +0000] [8] [DEBUG] 2 workers
[2022-01-02 15:37:04 +0000] [11] [DEBUG] POST /validate/ap/file
[2022-01-02 15:37:04 +0000] [11] [DEBUG] start validate file
endpoint
[2022-01-02 15:37:04 +0000] [11] [DEBUG] finish request to validate
file endpoint
```

UI log example

```
[2022-01-02 15:34:45 +0000] [11] [DEBUG] GET /
[2022-01-02 15:34:45 +0000] [11] [DEBUG] request index view
[2022-01-02 15:34:45 +0000] [11] [DEBUG] render index view
```

The RDF validator application uses the following environment variables to define logs location:

Description	Value	Associated variable
API logs	logs/api.log	RDF_VALIDATOR_APILOGS
UI logs	logs/ui.log	RDF_VALIDATOR_UILOGS
Celery logs	logs/celery.log	RDF_VALIDATOR_CELERY_LOGS

Table 4: RDF validator log configurations

8 Add a new application profile

There is an application profile already provided within the system that resides in `resource/aps` folder called `main` which contain the following shape files: `skosShapes.shapes.ttl`, `euvocShapes.shapes.ttl`, and `extensionShapes.shapes.ttl`. For adding a new application profile create a new folder under `resource/aps` with the name of your new application profile and add the shape files inside. It then should become available to the system; this can be checked by calling the API Application Profiles endpoint described here in chapter 10.1.

Folder structure of application profiles:

```
resources/
  aps/          <--- contains application profiles
    main/       <--- provided by default
      euvocShapes.shapes.ttl
      extensionShapes.shapes.ttl
      skosShapes.shapes.ttl
    alternative/ <--- example of another application profile
      other.shapes.ttl
      alternative.shapes.ttl
```

9 Custom templates

To configure the templates used for generating the custom validation reports you have to modify the currently available ones, found at `resource/templates`.

The templates are written in *Jinja2* templating language [3]. The data source access is facilitated through the *eds4jinja2* library [2]. If you are familiar with Jinja2 language a short introduction to how to use *eds4jinja2* is available on the documentation page¹. Also the default template can be seen as an example accessible in the repository².

Folder structure

```
resources/
  templates/
    html/      <--- custom html report
    json/      <--- custom json report
    query/     <--- queries used by the templates
      validator_query.rq
```

9.1 HTML template variant

Folder structure

```
html/
  config.json  <--- configuration file
  templates/   <--- jinja html templates
    sections/
```

¹<https://eds4jinja2.readthedocs.io/en/latest/>

²<https://github.com/meaningfy-ws/rdf-validator-ws/tree/master/resources/templates>

```
simple_view.html
layout.html
macros.html
main.html
```

Template structure

The HTML template is built by combining four major parts as layout, main, macros and sections. The layout file *layout.html* will have the rules of how the report will look like in terms of positioning and styling. Macros will contain all the jinja2 macros used across the template. A section represents the result of a query that was run with additional html and will be used to build the report. As the name suggests the main file of the html template is *main.html*. Here is where every other file that are a different section in the report are included and will form the HTML report.

Example of including a section in the main html file:

```
{% include "sections/simple_view.html" with context %}
```

Each section file has one or more variables where the SPARQL query result is saved as a pandas dataframe.

Example used in *simple_view.html*:

```
{% set content, error = from_rdf_file(conf.report_data_file).
    with_query_from_file(conf.query_file).fetch_tabular() %}
{% if not content.empty %}
    {% call mc.render_fetch_results(content, error) %}
        {% set compress_uris = simplify_uri_columns_in_tabular(
            data_frame=content, namespace_inventory=namespaces,
            error_fail=False) %}
        <section class="ui basic segment">
            {{ mc.pandas_table(content, "Validation issues") }}
        </section>
    {% endcall %}
{% endif %}
```

9.2 JSON template variant

Folder structure

```
json/
  config.json  <--- configuration file
```

```
templates/    <--- jinja json templates
  main.json
```

Note: make sure that in the templates folder there is a file named the same as the one defined in the *config.json* file (i.e "template": "main.json")

Template structure

The JSON report is automatically built by running all queries that are found in the *queries* folder as the system has autodiscover process for this. In the beginning of this report there will be 3 keys that will show the metadata of the report like dataset used, created time and application profile used. Each query result can be identified in the report by the filename and will contain a results key that will represent the result set brought back by the query.

10 API documentation

10.1 Get application profiles

Get application profiles names and their SHACL shapes files.

URL	ACTION
/aps	GET

Response

200

```
[
  {
    "application_profile": "main",
    "shapes_files": [
      "skosShapes.shapes.ttl",
      "euvocShapes.shapes.ttl",
      "extensionShapes.shapes.ttl"
    ]
  }
]
```

10.2 Get all validation reports

List the existent validation reports and the metadata about the validation.

URL	ACTION
/validations	GET

Response

200

```
[
  {
    "application_profile": "",
    "created_at": "09-Jan-2022T16:42:02",
    "custom_html_report": "custom.html",
    "custom_json_report": "custom.json",
    "file": "/usr/src/app/db/7d5a0396-7fbc-4d86-b84b-261f306f20a0/777a3696-637f-4f41-83f4-47b2062b83cbcourts-source-ap.rdf",
    "graphs": null,
    "html_report": "report.html",
    "shacl_shape_files": [
      "/usr/src/app/db/7d5a0396-7fbc-4d86-b84b-261f306f20a0/dfeaa507-6c93-4418-839b-1441d2aaa9bfskosShapes.shapes.ttl",
      "/usr/src/app/db/7d5a0396-7fbc-4d86-b84b-261f306f20a0/e1657ec9-88d2-4979-bb87-ccec352b570aevocShapes.shapes.ttl"
    ],
    "ttl_report": "report.ttl",
    "type": "file",
    "uid": "31ada0d2-b822-46b5-96b5-0d3c8d067ddf",
    "url": "",
    "zip_report": "report.zip"
  },
  {
    "application_profile": "main",
    "created_at": "09-Jan-2022T16:37:04",
    "custom_html_report": "custom.html",
    "custom_json_report": "custom.json",
    "file": "/usr/src/app/db/96e0e5db-7b82-4e64-aede-f65ce5a5f19b/6bf7bf64-c82c-4333-85a1-cbb395687625filetypes-source-ap.rdf",
  }
]
```

```

    "graphs": null,
    "html_report": "report.html",
    "shacl_shape_files": [
        "/usr/src/app/resources/aps/main/skosShapes.shapes.ttl",
        "/usr/src/app/resources/aps/main/euvocShapes.shapes.ttl",
        "/usr/src/app/resources/aps/main/extensionShapes.shapes.
        ttl"
    ],
    "ttl_report": "report.ttl",
    "type": "file",
    "uid": "c53a0f42-653f-43f8-9768-532dcada4b47",
    "url": "",
    "zip_report": "report.zip"
  }
]

```

10.3 Get validation report

Get specific HTML, JSON, TTL or ZIP (all report types in a zipped file) report.

URL	ACTION
/validations/{validation_id}	GET

Parameters

Name	Description
validation_id	validation unique id
report_type	type of report requested. accepted values: html, json, ttl and zip

Response

200

Report successfully retrieved.

Note: selecting the zip report extension the file will contain the following files:

- `custom.html` - custom HTML report generated by the template found in `resources/templates`.
- `custom.json` - custom JSON report generated by the template found in `resources/templates`.
- `report.html` - HTML report generated by RDFUnit
- `report.ttl` - TTL report generated by RDFUnit

422

```
{
  "detail": "Wrong report_extension format. Accepted formats: ttl,
            html, json, zip",
  "status": 422,
  "title": "Unprocessable Entity",
  "type": "about:blank"
}
```

10.4 Delete validation report

Delete specific validation

URL	ACTION
<code>/validations/{validation_id}</code>	DELETE

Parameters

Name	Description
<code>validation_id</code>	validation unique id

Response

200

```
{
```



```
{
  "message": "Validation with c53a0f42-653f-43f8-9768-532dcada4b47
             successfully removed"
}
```

404

```
{
  "detail": "Validation with c53a0f42-653f-43f8-9768-532dcada4b47
            doesn't exist",
  "status": 404,
  "title": "Not Found",
  "type": "about:blank"
}
```

10.5 Delete all validation reports

Delete all validations

URL	ACTION
/remove-reports	DELETE

Response

200

```
{
  "message": "Successfully removed all validations"
}
```

10.6 Validate file

Validate an RDF file with the provided SHACL shapes.

URL	ACTION
/validate/shapes/file	POST

Body*multipart/form-data*

Name	Required	Type	Description
<code>data_file</code>	true	file	The file to be validated.
<code>schema_file0</code>	- false	file	The content of the SHACL shape files defining the validation constraints.
<code>schema_file4</code>			

Response

200

```
{
  "task_id": "1dffa147-d25f-4a6e-a767-b7417b9c0b9e"
}
```

10.7 Validate file with Application Profile

Validate an RDF file with the provided Application Profile.

URL	ACTION
<code>/validate/ap/file</code>	POST

Body*multipart/form-data*

Name	Required	Type	Description
<code>data_file</code>	true	file	The file to be validated.

Table 15 continued from previous page

application_profile	true	string	The application profile selected for validation. Check section 10.1 to get available application profiles.
---------------------	------	--------	--

Response

200

```
{
  "task_id": "1dffa147-d25f-4a6e-a767-b7417b9c0b9e"
}
```

10.8 Validate SPARQL Endpoint

Validate a SPARQL endpoint with the provided SHACL shapes and optionally restricted to provided graphs.

URL	ACTION
/validate/shapes/url	POST

Body

multipart/form-data

Name	Required	Type	Description
sparql_endpoint_url	true	string	The endpoint to validate
graphs	false	array	An optional list of named graphs to restrict the scope of the validation
schema_file0 schema_file4	- false	file	The content of the SHACL shape files defining the validation constraints.

Table 17 continued from previous page

Response

200

```
{
  "task_id": "1dffa147-d25f-4a6e-a767-b7417b9c0b9e"
}
```

10.9 Validate SPARQL Endpoint with Application Profile

Validate a SPARQL endpoint with the provided Application Profile and optionally restricted to provided graphs.

URL	ACTION
/validate/ap/url	POST

Body*application/json*

Name	Required	Type	Description
sparql_endpoint_url	true	string	The endpoint to validate
graphs	false	array	An optional list of named graphs to restrict the scope of the validation
application_profile	true	string	The application profile selected for validation. Check section 10.1 to get available application profiles.

Response

200

```
{
  "task_id": "1dffa147-d25f-4a6e-a767-b7417b9c0b9e"
}
```

10.10 List active tasks

URL	ACTION
/tasks/active	GET

Response

200

```
[
  {
    "acknowledged": true,
    "args": [
      "b1a25374-0bf6-416b-94c3-c63c2e8fe827",
      "/usr/src/app/db/7767c52a-ccdc-400a-a927-41a70d745c48/8d0e2b10-a451-4bac-935c-7bd31d2ed274courts-source-ap.rdf",
      [
        "/usr/src/app/db/7767c52a-ccdc-400a-a927-41a70d745c48/1de6320f-fe6d-4d0b-9363-f485a42018e2skosShapes.shapes.ttl",
        "/usr/src/app/db/7767c52a-ccdc-400a-a927-41a70d745c48/ed9b67d1-dfe1-4880-8962-dc800eec06d6euvocShapes.shapes.ttl"
      ],
      "/usr/src/app/db/7767c52a-ccdc-400a-a927-41a70d745c48"
    ],
    "delivery_info": {
      "exchange": "",
      "priority": 0,
      "redelivered": null,
      "routing_key": "celery"
    },
    "hostname": "celery@1014fd21d778",
    "id": "dfb406f8-d59c-4c73-b332-47b7259e447e",
  }
]
```

```
[{"kwargs": {},  
  "name": "validate_file",  
  "time_start": 1641745019.7248263,  
  "type": "validate_file",  
  "worker_pid": 28  
}]
```

10.11 Get task status

Get specific task status

URL	ACTION
/tasks/{task_id}	GET

Parameters

Name	Description
task_id	task unique id

Response

200

```
{  
  "task_id": "dfb406f8-d59c-4c73-b332-47b7259e447e",  
  "task_result": null,  
  "task_status": "PENDING"  
}
```

10.12 Stop task execution

Contents

URL	ACTION
/tasks/{task_id}	DELETE

Parameters

Name	Description
task_id	task unique id

Response

200

```
{
  "message": "task 26651a0f-e8ae-490a-b500-ccf7b90080dd set for
    revoking."
}
```

406

```
{
  "detail": "task already finished executing or does not exist",
  "status": 406,
  "title": "Not Acceptable",
  "type": "about:blank"
}
```

References

- [1] E. Costetchi. Asset publication lifecycle architecture. Recommendation, Publications Office of the European Union, September 2020.
- [2] Meaningfy.ws. Embedded Datasource Specification in Jinja2 templates, 2020. URL <https://eds4jinja2.readthedocs.io/en/latest/>.
- [3] Pallets. Jinja 2 templating language, 2007. URL <https://jinja.palletsprojects.com/en/2.11.x/>.