Publications Office
of the European Union

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

Installation guide for the RDF fingeprinting service

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 fingeprinting 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	18 February 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

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 fingerprinter
2. Fingerprinter 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 port UI	HTTP port API
RDF fingerprinter	8020	4020
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

This describes how to setup the RDF fingerprinter application on for production level deployment. In order for the services to function properly a CentOS system with python version 3.6, Fuseki, and Redis service should be setup and running with the appropriate ports and addresses configured in the environment variable file. For information on how to setup the Fuseki service see the `https://jena.apache.org/documentation/fuseki2/`.

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

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

To start the services using the Makefile commands:

```
make install-python-dependencies
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

# run Celery
celery -A fingerprinter.adapters.celery.celery_worker worker --
    loglevel ${RDF_FINGERPRINTER_LOG_LEVEL} --logfile ${
    RDF_FINGERPRINTER_CELERY_LOGS} --detach

# run api server
gunicorn --timeout ${RDF_FINGERPRINTER_GUNICORN_TIMEOUT-1200} --
    workers ${RDF_FINGERPRINTER_GUNICORN_API_WORKERS-2} --bind
    0.0.0.0:${RDF_FINGERPRINTER_API_PORT} --reload --log-file ${
    RDF_FINGERPRINTER_API_LOGS} --log-level ${
    RDF_FINGERPRINTER_LOG_LEVEL} fingerprinter.entrypoints.api.run:
    app --daemon

# run ui server
gunicorn --timeout ${RDF_FINGERPRINTER_GUNICORN_TIMEOUT-1200} --
    workers ${RDF_FINGERPRINTER_GUNICORN_UI_WORKERS-1} --bind
    0.0.0.0:${RDF_FINGERPRINTER_UI_PORT} --reload --log-file ${
    RDF_FINGERPRINTER_UI_LOGS} --log-level ${
```

```
    RDF_FINGERPRINTER_LOG_LEVEL} fingerprinter.entrypoints.ui.run:
    app --daemon
```

To stop the services run

```
source env/bin/activate

pkill -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.

The following applications have to be available in the system:

```
        sudo yum install redis curl
```

You can also find the required packages in the `install_os_dependencies.sh` file.

7.1 RDF fingerprinter

The RDF fingerprinter is an online platform that provides the possibility to fingerprint an RDF file or SPARQL endpoint. The RDF fingerprinter API is the core service providing the RDF fingerprinting functionality. The URL and port are described below, as well as the request timeout:

Description	Value	Associated variable
Service URL	http://localhost	RDF_FINGERPRINTER_API_LOCATION
Service API port	4020	RDF_FINGERPRINTER_API_PORT
Is in debug mode	False	RDF_FINGERPRINTER_DEBUG
Logging level	DEBUG	RDF_FINGERPRINTER_LOG_LEVEL
Service UI port	8020	RDF_FINGERPRINTER_UI_PORT
Web server worker process timeout	1200	RDF_FINGERPRINTER_GUNICORN_TIMEOUT

Table 2: RDF fingerprinter configurations

Note: Possible values for logging levels are explained in more detail docs.python.org/3.6/library/logging.html :

- DEBUG
- INFO
- WARNING
- ERROR
- CRITICAL

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 fingerprinter project it serves the purpose of enabling multiprocessing of fingerprinting report generation.

The RDF fingerprinter application uses the following Celery environment variables

Description	Value	Associated variable
Redis location	`redis://localhost`	RDF_FINGERPRINTER_REDIS_LOCATION
Redis port	6379	RDF_FINGERPRINTER_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 fingerprinting is presented below:

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

CELERY_FINGERPRINT_FILE = 'fingerprint_file'


@celery_worker.task(name=CELERY_FINGERPRINT_FILE, bind=True)
def async_fingerprint_file(self, file_path: str,
   db_cleanup_location: str, graphs: list = None):
        ...
```

7.3 Configure and read logs

Every service provided by the RDF fingerprinter 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.

`logs/celery.log` contains the logs of the Celery workers. This contains the logs of the tasks that are being processed; here is where you would do most of the log hunting to debug inconsistent behavior.

API log example

```
[2022-02-17 15:26:16 +0000] [1] [INFO] Starting gunicorn 20.1.0
[2022-02-17 15:26:16 +0000] [1] [DEBUG] Arbiter booted
[2022-02-17 15:26:16 +0000] [1] [INFO] Listening at: http
   ://0.0.0.0:4020 (1)
[2022-02-17 15:26:16 +0000] [1] [INFO] Using worker: sync
[2022-02-17 15:26:16 +0000] [9] [INFO] Booting worker with pid: 9
[2022-02-17 15:26:16 +0000] [10] [INFO] Booting worker with pid: 10
[2022-02-17 15:26:16 +0000] [1] [DEBUG] 2 workers
[2022-02-17 15:26:55 +0000] [9] [DEBUG] getting fingerprinting
   reports
```

UI log example

```
[2022-02-17 15:30:24 +0000] [10] [DEBUG] request index view
```

```
[2022-02-17 15:31:25 +0000] [10] [DEBUG] request fingerprint sparql
    endpoint view
[2022-02-17 15:31:25 +0000] [10] [DEBUG] return fingerprint sparql
   endpoint clean view
```

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

Description	Value	Associated variable
API logs	logs/api.log	RDF_FINGERPRINTER_API_LOGS
UI logs	logs/ui.log	RDF_FINGERPRINTER_UI_LOGS
Celery logs	logs/celery.log	RDF_FINGERPRINTER_CELERY_LOGS

Table 4: RDF fingerprinter 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 **??**.

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 fingerprinting reports you have to modify the currently available ones, found at `resources/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 page[1]. Also the default template can be seen as an example accessible in the repository[2].

Folder structure

```
resources/
    templates/
      html/      <--- custom html report
      json/      <--- custom json report
```

9.1 HTML and JSON template variant

HTML Folder structure

```
html/
  config.json   <--- configuration file
  templates/    <--- jinja html templates
    macros/
      builders.html  <--- macro for building the report
      renderers.html <--- macro for rendering information inside
         the report
      selectors.html <--- macro for selecting information from the
         report
    layout.html  <--- layout of the report
    main.html    <--- "start" of the template
    prefixes.json  <--- prefixes used in the report
```

Note: the `json` report structure is similar to the `HTML`

JSON Folder structure

[1]`https://eds4jinja2.readthedocs.io/en/latest/`

[2]`https://github.com/meaningfy-ws/rdf-fingerprinter-ws/tree/main/resources/templates`

```
json/
  config.json   <--- configuration file
  templates/    <--- jinja json templates
    macros/
      builders.json  <--- macro for building the report
      renderers.json <--- macro for rendering information inside
         the report
      selectors.json <--- macro for selecting information from the
         report
    layout.json  <--- layout of the report
    main.json    <--- "start" of the template
    prefixes.json  <--- prefixes used in the report
```

10 API documentation

10.1 Get all fingerprinting reports

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

URL	ACTION
/reports	GET

Response

200

```
[
  {
      "created_at": "14-Feb-2022T13:28:09",
      "file": "/usr/src/app/db/48605058-4795-492b-9793-1074201d9af8
         /090f9ccf-e2bb-4ff1-995a-aa65a2fb9defcontinents-source-ap.
         rdf",
      "fingerprinting_on": "file",
      "graphs": [],
      "html_report": "report.html",
      "json_report": "report.json",
      "uid": "e876dc23-0e31-46a5-b5f3-102db8b15c11",
      "url": "",
      "zip_report": "report.zip"
  },
]
```

10.2 Get fingerprinting report

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

URL	ACTION
`/reports/{fingerprinting_id}`	`GET`

Parameters

Name	Description
`fingerprinting_id`	fingerprinting unique id
`report_type`	type of report requested. accepted values: html, json, and zip

Response

`200`

Report successfully retrieved.

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

- `report.html` - custom HTML
- `report.json` - custom JSON

`422`

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

The flow to retrieve a fingerprinting report is as follows:

1. send a request to on of the fingerprinting endpoints
2. use the UID provided in the response to retrieve the report by calling the `/reports/{UID}?report_type={type}` endpoint

Note: This request just retrieves the reports, it doesn't regenerate them. If the original data used in the fingerprinting suffered modifications, step 1. has to be run again.

10.3 Delete fingerprinting report

Delete specific fingerprinting

URL	ACTION
`/reports/{fingerprinting_id}`	`DELETE`

Parameters

Name	Description
`fingerprinting_id`	fingerprinting unique id

Response

200

```
{
  "message": "Fingerprinting with bbc32fb7-c20c-45f3-9949-
     d294d1755870 successfully removed"
}
```

404

```
{
  "detail": "Fingerprinting with bbc32fb7-c20c-45f3-9949-
     d294d1755870 doesn't exist",
  "status": 404,
  "title": "Not Found",
  "type": "about:blank"
}
```

10.4 Delete all fingerprinting reports

Delete all fingerprinting

URL	ACTION
`/remove-reports`	`DELETE`

Response

200

```
{
  "message": "All fingerprintings successfully removed"
}
```

10.5 Fingerprint file

Fingerprint an RDF file.

URL	ACTION
`/fingerprint-file`	`POST`

Body

multipart/form-data

Name	Required	Type	Description
`data_file`	true	`file`	The file to be fingerprinted.

Table 12 continued from previous page

Response

200

```
{
  "task_id": "2383dd04-5dfc-4523-9f6b-e9304495a4fc"
}
```

10.6 Fingerprint SPARQL endpoint

Fingerprint SPARQL endpoint and optionally restricted to provided graphs.

URL	ACTION
`/fingerprint-sparql-endpoint`	`POST`

Body

multipart/form-data

Name	Required	Type	Description
`sparql_endpoint_url`	true	`string`	The endpoint to fingeprint
`graphs`	false	`array`	An optional list of named graphs to restrict the scope of the fingerprinting

Response

200

```
{
  "task_id": "e4ae3a48-4ecd-4ed8-8b3e-6445f106d4b2"
}
```

10.7 List active tasks

URL	ACTION
`/tasks/active`	`GET`

Response

200

```
  [
    {
        "acknowledged": true,
        "args": [],
        "delivery_info": {
            "exchange": "",
            "priority": 0,
            "redelivered": null,
            "routing_key": "celery"
        },
        "hostname": "celery@a2b79d7ffdcb",
        "id": "79e95946-94b4-45c1-8090-54dbb43dd490",
        "kwargs": {
            "db_cleanup_location": "/usr/src/app/db/ac921023-ac15-4
               a25-85e1-b33b78674e02",
            "file_path": "/usr/src/app/db/ac921023-ac15-4a25-85e1-
               b33b78674e02/f4c0e71d-c039-4396-8b22-55
               cd63ad2e0ecourts-source-ap.rdf"
        },
        "name": "fingerprint_file",
        "time_start": 1645204152.7428734,
        "type": "fingerprint_file",
        "worker_pid": 27
    }
]
```

Note: the brief explanation of fields.

- `id` - ID of the task.
- `name` - name of the task, in this case, it is the same as the task `type`.
- `args` - positional arguments passed to the task.
- `kwargs` - keywords arguments passed to the task.

- `type` - type of the task.
- `hostname` - hostname of the worker that is running the task.
- `acknowledged` - True when task was acknowledged by the broker.
- `delivery_info`
  - `exchange` - name of the exchange the task was published to
  - `priority` - priority of the task
  - `redelivered` - True when task was redelivered by the broker
  - `routing_key` - routing key of the task
- `time_start` - time of the task start.
- `worker_pid` - PID of the worker that is executing the task.

10.8 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"
}
```

Note: Task statuses and their meanings::

- `FAILURE` - Task failed.
- `PENDING` - Task state is unknown or the task doesn't exist.
- `REVOKED` - Task was revoked.
- `STARTED` - Task was started by a worker.
- `SUCCESS` - Task succeeded.

10.9 Stop task execution

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/`.