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

Installation guide for the RDF-Diffing service using docker

Disclaimer

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-Diffing service using docker
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	12 December 2021
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 6
2 Scope 6
3 Target audience 6
4 Technology background 7
5 Requirements 8
6 Installation 8
7 Configuration 9
7.1 RDF differ 11
7.2 Celery worker and Redis 11
7.3 RDF differ dedicated triple store 12
8 Add a new application profile template 13
8.1 HTML template variant 13
8.2 JSON template variant 18
9 API documentation 20
9.1 Get application profiles 20
9.2 Get dataset deltas 20
9.3 Get dataset delta 22
9.4 Delete dataset delta 23
9.5 Create dataset delta 24
9.6 Create a report 25
9.7 Get report 26
9.8 List active tasks 27
9.9 Get task status 28
9.10 Stop task execution 29
10 SPARQL Queries 30
10.1 Change type inventory 30
10.2 Naming conventions 32

10.3 Structure 32
10.4 Example of diffing query 35
Appendices 38

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 Docker images [2], which is motivated in Section 4.

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

2 Scope

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

1. RDF differ
2. Celery worker
3. Fuseki triplestore [4]
4. Redis

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 the *Docker technology* [2, 3]. Docker is a set of platform as a service (PaaS) products that use OS-level virtualisation to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore collectively, use fewer resources than virtual machines.

Docker technology is chosen because it solves the problem known in the system administration world as the "dependency hell", which refers to three specific issues: conflicting dependencies, missing dependencies, and platform differences.

Docker solved these issues by providing the means for imagines to package an application along with all of its dependencies easily and then run it smoothly in disparate development, test and production environments.

Docker Compose is a tool for defining and running multi-container Docker applications or application suites. It uses YAML files to configure the application's services and performs the creation and start-up and shutdown process of all the containers with a single command. The docker-compose command line interface (CLI) utility allows users to run commands on multiple containers at once, for example, building images, scaling containers, running containers that were stopped, and more. Commands related to image manipulation, or user-interactive options, are not relevant in Docker Compose because they address one container. The *docker-compose.yml* file is used to define an application's services and includes various configuration options.

The services and applications enumerated in Section 2 are packaged into Docker images. The associated docker-compose.yml file defines the suite of applications and micro-service configurations in order to be deployed and ran together with ease. This manual explains how to run and configure this suite of Docker containers using Docker Compose tool.

5 Requirements

Although Docker can be executed on any platform, for performance and security reasons we recommend using a Linux OS with kernel version 5.4x or higher. The services have been tested on Ubuntu 20 server.

There is a range of ports that must be available on the host machine as they will be bound to by different docker 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 differ	8030	4030
dedicated Fuseki		3030
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 to run the services it is necessary to have Docker server and docker-compose tool installed. To install them following the instructions provided at the following locations

1. Docker - `https://docs.docker.com/engine/install`
2. Docker Compose - `https://docs.docker.com/compose/install`

In case you are using Debian-like OS such as Ubuntu, you may simply run the following Bash commands to install and set the appropriate permissions.

```
sudo apt -y install docker.io docker-compose git make
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

Next, copy the rdf differ zip on the system you intend to run it and unzip it.

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

Downloading the Docker images will be triggered automatically on first request to start the services.

To start the services using Makefile

```
make start-services
```

To stop the services using Makefile

```
make stop-services
```

To start services without Makefile:

```
docker-compose --file docker/docker-compose.yml --env-file docker/.
    env up -d
```

To stop the services run

```
docker-compose --file docker/docker-compose.yml --env-file docker/.
    env down
```

7 Configuration

The deployment suite of micro-services is defined docker-compose.yml file. 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 docker-compose, a tool designed especially for managing multi-container Docker applications, by describing them in a single file. Then, with a single command, you create and build, start or stop all the services using that configuration file.

In order to avoid hard coding parameters, docker-compose allows you to define them externally. You have the option to define them as operating system level environment variables or provide them in a single file, which is passed as a parameter to the docker-compose tool using the *–env-file* command line argument. Having them in a single file makes much more 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 file is usually named *.env* and contains all of the parameters that you want to be able to change and that you need to build and run the defined containers.

Having the parameters in an *.env* file is very useful in a multitude of scenarios, where you would want to have different configurations for different environments where you might want to deploy. As a more specific example, consider a continuous delivery pipeline and the URLs and ports you want your containers to bind (or to connect) to. You thus can easily have two *.env* files, one named *test.env* and one named *acceptance.env*. Each file would have the same declared variables, but with different values for each of the continuous delivery pipeline stage where it's being deployed. The benefit is that you deploy and test/use the same containers/artifacts and are able to configure them, on the spot, according to the environment that they are integrated with.

Let's take, for example, the RDF Differ user interface Docker container, which is defined, in the *docker-compose.yml* file as it follows:

```
    rdf-differ-ui:
        container_name: rdf-differ-ui
        image: meaningfy/rdf-differ-ui:latest
        ports:
                - ${RDF_DIFFER_UI_PORT}:${RDF_DIFFER_UI_PORT}
        env_file: .env
        restart: always
        networks:
                - mydefault
```

The variable used in the definition of this service is just one, ***RDF_DIFFER_UI_PORT***.

And the place where docker-compose will look for that variable is specified in the ***env_file: .env*** line.

Now, if you look in the ".env" file, you will quickly see that the variable is defined as ***RDF_DIFFER_UI_PORT=8030***. Change the value of the port, rebuild the micro-services and RDF Differ will no longer be listening on 8030, but on the new port that you specified.

This section describes the important configurations options available for each of the services.

7.1 RDF differ

The RDF differ application exposes an API and an UI and depends on a dedicated triple store. the RDF diff API is the core service providing the RDF diffing functionality. The URL and port are described below, as well as the request timeout:

Description	Value	Associated variable
Service URL	http://rdf-differ-api	RDF_DIFFER_API_LOCATION
Service API port	4030	RDF_DIFFER_API_PORT
Is in debug mode	True	RDF_DIFFER_DEBUG
Service UI port	8030	RDF_DIFFER_UI_PORT
Web server worker process timeout	1200	RDF_DIFFER_GUNICORN_TIMEOUT

Table 2: RDF differ configurations

Please note that the domain specified in in the URL is only available inside the Docker network and is not visible from the outside. Its purpose is to provide a named way for a service to connect to another service.

7.2 Celery worker and Redis

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 differ project it serves the purpose of enabling multiprocessing of both diff creation and report generation.

Redis is used as celery's backend to store information about tasks. Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. Redis provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams.

A docker configuration is provided for both celery and redis (`celery-worker` and `redis`)

The RDF differ application uses the following Celery environment variables

Description	Value	Associated variable
Redis location	`redis://redis`	RDF_DIFFER_REDIS_LOCATION
Redis port	6379	RDF_DIFFER_REDIS_PORT

Table 3: Celery environment configurations

7.3 RDF differ dedicated triple store

RDF differ depends on a Fuseki triple store to calculate and persist the diffs. The available configurations are described below.

Description	Value	Associated variable
Admin account password	admin	RDF_DIFFER_FUSEKI_ADMIN_PASSWORD
User name	admin	RDF_DIFFER_FUSEKI_USERNAME
Password	admin	RDF_DIFFER_FUSEKI_PASSWORD
Folder where Fuseki stores data	./data/diff	RDF_DIFFER_FUSEKI_DATA_FOLDER
External port	3030	RDF_DIFFER_FUSEKI_PORT
Internal port	3030	
Additional arguments passed to JVM	-Xmx2g	RDF_DIFFER_FUSEKI_JVM_ARGS
URL	http://rdf-differ-fuseki	RDF_DIFFER_FUSEKI_LOCATION

Table 4 continued from previous page

Table 4: RDF differ dedicated triple store configurations

8 Add a new application profile template

There are several application profiles already provided within the system that resides in `resource/templates` folder. For adding a new application profile create a new folder under `resource/templates` with the name of your new application profile and following the structure explained below.

Folder structure needed for adding a new application profile:

```
templates/
  diff_report/
  new_application_profile/
   queries/      <--- contains SPARQL queries
    query1.rq
    query2.rq
   template_variants/
    html/        <--- contains files for a html template
    json/        <--- contains files for a json template
```

8.1 HTML template variant

Folder structure

```
html/
  config.json   <--- configuration file
  templates/    <--- jinja html templates
    file1.html
    file2.html
    main.html
```

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.html"`)

Template structure

The HTML template is built be 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 suggest 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 "conceptscheme/added_instance_concept_scheme.html" with
   context %}
```

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

Example:

```
{% set content, error = from_endpoint(conf.default_endpoint).
   with_query_from_file(conf.query_files["added_instance_concept.rq
   "]).fetch_tabular() %})
```

Note: the system has in place an autodiscover process for the SPARQL queries in the queries folder. Make sure that the file name added for the variable above (*added_instance_concept.rq*) exists in the queries folder.

Adjusting an existing template

Adding a new query/section

To add a query a new file needs to be created and added into the queries folder as the system will autodiscover this. After this is done a new html file that will represent a new section needs to be created. The content of this is similar to the existing ones and the only thing that needs to be adjusted will be the query file name in the content variable definition as presented below:

```
{% set content, error = from_endpoint(conf.default_endpoint).
   with_query_from_file(conf.query_files["new_query_file.rq"]).
   fetch_tabular() %})
```

As a final step, the created html file needs to be included in the report and to do this it has to be included in the main.html file by using the include block.

```
--- relative path to the new html path
{% include "conceptscheme/labels/new file name.html" with context
   %}
```

For adding a count query that will be used in the statistics section the steps are a bit different. First, will need to add the new query file following the naming conventions and adding the prefix `count_` to the file name in queries folder. After this, the statistics.html will need to be modified as follows:

1. Create a new row in the existing table by using `<tr>` tag.
2. Create the necessary columns for the newly created row. Each row should have 7 values as this is the defined table structure *(Property group, Property, Added, Deleted, Updated, Moved, Changed)* and each of them should be included by using a `<td>` tag if you are not using the block below to autogenerate this.

```
{% set content, error = from_endpoint(conf.default_endpoint).
   with_query_from_file(conf.query_files["new_count_query_file_name
   .rq"]).fetch_tabular() %}

{% call mc.render_fetch_results(content, error) %}
  {{ mc.count_value(content) }}
{% endcall %}
```

Note: the order of the cells is important. If you don't want to include a type of operation just create a `<td>` with a desired value (i.e `<td>N/A</td>`). To avoid confusions, count queries should be added for all type of operations. The example below will show how to add a complete row in the statistics section of the report:

```
<tr>
  <td>Name of the property group</td>
  <td>Name of the property</td>
  --- this will bring the number generated from the SPARQL query
     for added occurences and will create the <td> tag
      {% set content, error = from_endpoint(conf.default_endpoint).
         with_query_from_file(conf.query_files["
         count_added_property_concept_scheme_pref_label.rq"]).
         fetch_tabular() %}

      {% call mc.render_fetch_results(content, error) %}

      {{ mc.count_value(content) }}
      {% endcall %}

  --- this will bring the number generated from the SPARQL query
     for deleted occurences and will create the <td> tag
      {% set content, error = from_endpoint(conf.default_endpoint).
         with_query_from_file(conf.query_files["
         count_deleted_property_concept_scheme_pref_label.rq"]).
         fetch_tabular() %}
```

```
      {% call mc.render_fetch_results(content, error) %}

      {{ mc.count_value(content) }}
      {% endcall %}

  --- this will bring the number generated from the SPARQL query
     for updated occurences and will create the <td> tag
      {% set content, error = from_endpoint(conf.default_endpoint).
         with_query_from_file(conf.query_files["
         count_updated_property_concept_scheme_pref_label.rq"]).
         fetch_tabular() %}

      {% call mc.render_fetch_results(content, error) %}

      {{ mc.count_value(content) }}
      {% endcall %}

  --- this will bring the number generated from the SPARQL query
     for moved occurences and will create the <td> tag
      {% set content, error = from_endpoint(conf.default_endpoint).
         with_query_from_file(conf.query_files["
         count_moved_property_concept_scheme_pref_label.rq"]).
         fetch_tabular() %}

      {% call mc.render_fetch_results(content, error) %}

      {{ mc.count_value(content) }}
      {% endcall %}

  --- this will bring the number generated from the SPARQL query
     for changed occurences and will create the <td> tag
      {% set content, error = from_endpoint(conf.default_endpoint).
         with_query_from_file(conf.query_files["
         count_changed_property_concept_scheme_pref_label.rq"]).
         fetch_tabular() %}

      {% call mc.render_fetch_results(content, error) %}

      {{ mc.count_value(content) }}
      {% endcall %}

</tr>
```

Removing a query/section

To remove a section from the existing report you just need to delete or comment

the include statement from the main.html file. If you decide to delete the include statement it's recommended to delete the query from the queries folder as well to avoid confusions later on.

Example of include statement to remove:

```
{% include "conceptscheme/labels/
   added_property_concept_scheme_pref_label.html" with context %}
```

To remove a row from the statistics section just delete or comment the `<tr>` block from the *statistics.html* file:

```
<tr>
  <td>Labels</td>
  <td>skos:prefLabel</td>

      {% set content, error = from_endpoint(conf.default_endpoint).
         with_query_from_file(conf.query_files["
         count_added_property_concept_scheme_pref_label.rq"]).
         fetch_tabular() %}

      {% call mc.render_fetch_results(content, error) %}

      {{ mc.count_value(content) }}
      {% endcall %}


      {% set content, error = from_endpoint(conf.default_endpoint).
         with_query_from_file(conf.query_files["
         count_deleted_property_concept_scheme_pref_label.rq"]).
         fetch_tabular() %}

      {% call mc.render_fetch_results(content, error) %}

      {{ mc.count_value(content) }}
      {% endcall %}


      {% set content, error = from_endpoint(conf.default_endpoint).
         with_query_from_file(conf.query_files["
         count_updated_property_concept_scheme_pref_label.rq"]).
         fetch_tabular() %}

      {% call mc.render_fetch_results(content, error) %}
```

```
      {{ mc.count_value(content) }}
      {% endcall %}


      {% set content, error = from_endpoint(conf.default_endpoint).
         with_query_from_file(conf.query_files["
         count_moved_property_concept_scheme_pref_label.rq"]).
         fetch_tabular() %}

      {% call mc.render_fetch_results(content, error) %}

      {{ mc.count_value(content) }}
      {% endcall %}


      {% set content, error = from_endpoint(conf.default_endpoint).
         with_query_from_file(conf.query_files["
         count_changed_property_concept_scheme_pref_label.rq"]).
         fetch_tabular() %}

      {% call mc.render_fetch_results(content, error) %}

      {{ mc.count_value(content) }}
      {% endcall %}
</tr>
```

8.2 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:

```
{
  --- Metadata

  "dataset_name": "name of dataset",
  "timestamp": "time",
  "application_profile": "application profile namme",

  --- Query result set

  "count_changed_property_concept_definition.rq":
  {
      "head":
      {
          "vars":
          [
              "entries"
          ]
      },
      "results":
      {
          "bindings":
          [
              {
                  "entries":
                  {
                      "datatype": "http://www.w3.org/2001/XMLSchema
                         #integer",
                      "type": "literal",
                      "value": "0"
                  }
              }
          ]
      }
  }
}
```

Adjusting an existing template

Removing a query/section To remove a query result set from the report simply remove the query from the queries folder.

Note: doing this will also affect the HTML template and it's recommended to adjust

the html template, if this exists as a template variant for the application profile that you are working with, following the instruction above to avoid errors when generating the HTML template variant.

9 API documentation

9.1 Get application profiles

Get application profiles names and their template variations.

URL	ACTION
`/aps`	`GET`

Response

200

```
[
  {
    "application_profile": "skos-core-en-only",
    "template_variations": [
      "html",
      "json"
    ]
  },
  {
    "application_profile": "skos-core-lang-fallback",
    "template_variations": [
      "html",
      "json"
    ]
  }
]
```

9.2 Get dataset deltas

List the existent dataset deltas

URL	ACTION
/diffs	GET

Response

200


```
[
  {
      "current_version_graph": "http://publications.europa.eu/
         resource/authority/data-theme/version/new",
      "dataset_description": null,
      "dataset_id": "/diff18H35CGpD",
      "dataset_uri": "http://publications.europa.eu/resource/
         authority/data-theme/",
      "dataset_versions": [
          "new1",
          "old1"
      ],
      "diff_date": null,
      "new_version_id": "new",
      "old_version_id": "old",
      "query_url": "http://fuseki:3030/diff18H35CGpD/sparql",
      "version_history_graph": "http://publications.europa.eu/
         resource/authority/data-theme/version",
      "version_named_graphs": [
          "http://publications.europa.eu/resource/authority/data-
             theme/version/new",
          "http://publications.europa.eu/resource/authority/data-
             theme/version/old"
      ]
  },
  {
      "current_version_graph": "http://publications.europa.eu/
         resource/authority/data-theme/version/new",
      "dataset_description": null,
      "dataset_id": "/diff2F4ZLMgNu",
      "dataset_uri": "http://publications.europa.eu/resource/
         authority/data-theme/",
      "dataset_versions": [
          "new1",
          "old1"
      ],
      "diff_date": null,
      "new_version_id": "new",
```

```
      "old_version_id": "old",
      "query_url": "http://fuseki:3030/diff2F4ZLMgNu/sparql",
      "version_history_graph": "http://publications.europa.eu/
         resource/authority/data-theme/version",
      "version_named_graphs": [
          "http://publications.europa.eu/resource/authority/data-
             theme/version/new",
          "http://publications.europa.eu/resource/authority/data-
             theme/version/old"
      ]
  }
]
```


9.3 Get dataset delta

Get specific dataset delta

URL	ACTION
`/diffs/{dataset_id}`	`GET`

Parameters

Name	Description
`dataset_id`	dataset unique name

Response

200


```
{
  "current_version_graph": "http://publications.europa.eu/resource/
     authority/data-theme/version/new",
  "dataset_description": null,
  "dataset_id": "/diff18H35CGpD",
  "dataset_uri": "http://publications.europa.eu/resource/authority/
     data-theme/",
  "dataset_versions": [
    "new1",
```

```
    "old1"
  ],
  "diff_date": null,
  "new_version_id": "new",
  "old_version_id": "old",
  "query_url": "http://fuseki:3030/diff18H35CGpD/sparql",
  "version_history_graph": "http://publications.europa.eu/resource/
     authority/data-theme/version",
  "version_named_graphs": [
    "http://publications.europa.eu/resource/authority/data-theme/
       version/new",
    "http://publications.europa.eu/resource/authority/data-theme/
       version/old"
  ]
}
```


404


```
{
  "detail": "<datasetname> does not exist.",
  "status": 404,
  "title": "Not Found",
  "type": "about:blank"
}
```


9.4 Delete dataset delta

Delete specific dataset delta

URL	ACTION
`/diffs/{dataset_id}`	`DELETE`

Parameters

Name	Description
`dataset_id`	dataset unique name

Response

200

```
"<datasetname> deleted successfully."
```

404

```
{
  "detail": "<datasetname> does not exist.",
  "status": 404,
  "title": "Not Found",
  "type": "about:blank"
}
```

9.5 Create dataset delta

Create a diff delta

URL	ACTION
`/diffs`	`POST`

Body

multipart/form-data

Name	Required	Type	Description
`dataset_id`	true	`string`	The dataset identifier. This should be short alphanumeric string uniquely identifying the dataset.
`dataset_description`	true	`string`	The dataset description. This is a free text description fo the dataset.
`dataset_uri`	true	`string`	The dataset URI. For SKOS datasets this is usually the ConceptSchema URI.

Table 12 continued from previous page

`old_version_id`	true	`string`	Identifier for the older version of the dataset.
`new_version_id`	true	`string`	Identifier for the newer version of the dataset.
`old_version_file_content`	true	`file`	The content of the old version file.
`new_version_file_content`	true	`file`	The content of the new version file.
`new_version_id`	true	`string`	Identifier for the newer version of the dataset.

Response

200

```
{
  "dataset_name": "diff2F4ZLMgNu",
  "task_id": "cee03499-41b2-41e4-ae75-95b9383eea0c"
}
```

9.6 Create a report

URL	ACTION
`/diffs/reports`	`POST`

Body

application/json

Name	Required	Type	Description
dataset_id	true	string	The dataset identifier. This should be short alphanumeric string uniquely identifying the dataset.
application_profile	true	string	The application profile identifier
template_type	true	string	The template type identifier
rebuild	false	string	Flag to signal rebuilding the report even if already exists. ("true" or "false")

Response

200

```
{
  "application_profile": "skos-core-en-only",
  "task_id": "3cf43787-18e8-4927-aa5f-198da6b8bba2"
}
```

9.7 Get report

URL	ACTION
/diffs/report	GET

Parameters

Name	Description
dataset_id	dataset unique name
application_profile	The application profile identifier
template_type	The template type identifier

Response

200

Report file in specified format

9.8 List active tasks

URL	ACTION
/tasks/active	GET

Response

200


```
[
  {
    "acknowledged": true,
    "args": [
        "diff2F4ZLMgNu",
        "skos-core-en-only",
        "html",
        "/usr/src/app/reports",
        "/usr/src/app/resources/templates/skos-core-en-only/
           template_variants/html",
        {
            ...
        },
        {
            "available_reports": [
                {
                    "application_profile": "diff_report",
                    "template_variations": [
                        "html"
                    ]
                }
            ],
            "current_version_graph": "http://publications.europa.eu
               /resource/authority/data-theme/version/new1",
            "dataset_description": null,
            "dataset_id": "diff2F4ZLMgNu",
            "dataset_uri": "http://publications.europa.eu/resource/
               authority/data-theme/",
```

```
            "dataset_versions": [
                "new1",
                "old1"
            ],
            "diff_date": null,
            "new_version_id": "old1",
            "old_version_id": "new1",
            "query_url": "http://fuseki:3030/diff2F4ZLMgNu/sparql",
            "version_history_graph": "http://publications.europa.eu
               /resource/authority/data-theme/version",
            "version_named_graphs": [
                "http://publications.europa.eu/resource/authority/
                   data-theme/version/new1",
                "http://publications.europa.eu/resource/authority/
                   data-theme/version/old1"
            ]
        }
    ],
    "delivery_info": {
        "exchange": "",
        "priority": 0,
        "redelivered": null,
        "routing_key": "celery"
    },
    "hostname": "celery@e5d79cc45ba2",
    "id": "3cf43787-18e8-4927-aa5f-198da6b8bba2",
    "kwargs": {},
    "name": "generate_report",
    "time_start": 1638810064.448214,
    "type": "generate_report",
    "worker_pid": 25
  }
]
```


9.9 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": "6ce77efc-667e-4cf4-bcec-cc7870fcc2db",
  "task_result": true,
  "task_status": "SUCCESS"
}
```

9.10 Stop task execution

URL	ACTION
/tasks/{task_id}	DELETE

Parameters

Name	Description
task_id	task unique id

Response

200

```
{
  "message": "task 3cf43787-18e8-4927-aa5f-198da6b8bba2 set for
     revoking."
}
```

406

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

10 SPARQL Queries

10.1 Change type inventory

This section provides a change type inventory along with the patterns captured by each change type. We model the change as state transition operator between old (on the left) and teh new (on the right). The transition operator is denoted by the arrow symbol (→) On each sides of the transition operator, we use a compact notation following SPARQL triple patterns.

We use a set of conventions for each variable in the triple pattern, ascribing meaning to each of them and a few additional notations. These conventions are presented in the table below.

Notation	Meaning	Example
triple pattern `< _s p o_ >`	each item in the triple represents a SPARQL variable or an URI. For brevity we omit the question mark prefix (?) otherwise the SPARQL reading shall apply.	`i p v`
arrow →	state transition operator (from one version to the next)	`i1 p o` → `i2 p o`
`i` - in the triple pattern	the instance subject (assuming class instantiation)	`i p v`
`p` - in the triple pattern	the main predicate	`i p v`
`op` - in the triple pattern	the secondary predicate in a property chain (/)	`i p/op v`
`v` - in the triple pattern	the value of interest, which is object of the main or secondary predicate	`i p v`
slash (/)	the property chaining operation.	`p1/p2/p3/p4`

Table 22 continued from previous page

number (#)	the numeric suffixes help distinguish variables of the same type operation.	i1 p1 o1 → i2 p2 o2
zero (0)	denotes empty set or not applicable	0

The table below presents the patterns of change likely to occur in the context of maintaining SKOS vocabularies, but the abstraction proposed here may be useful way beyond this use case. The table represents a power product between the four types of change relevant to the current diffing context and the possible triple patterns in which they can occur. Cells that are marked with zero (0) mean that no check shall be performed for such a change type as it is included in onw of its siblings. The last two columns indicate whether quantification assumptions apply on either side of the transition operator.

change type	instance	property value	reified property value
Addition	0 → i	0 → i p v	0 → i p/op v
Deletion	i → 0	i p v → 0	i p/op v → 0
Value update	0	i p v1 → i p v2	i p/op v1 → i p/op v2
Movement (cross instance)	0	i1 p v → i2 p v	i1 p/op v → i2 p/op v
Movement (cross property)	0	i p1 v → i p2 v	i p1/op v → i p2/op v

The state transition patterns presented in the table above can be translated to SPARQL queries.

10.2 Naming conventions

Operations

Looking at patterns of change likely to occur in the context of maintaining SKOS vocabularies or to find in a diffing context we've identified 5 change types and mapped them for easy referencing as follows:

- Addition → Added
- Deletion → Deleted
- Value update → Updated
- Movement (cross property) → Changed
- Movement (cross instance) → Moved

Query file name

In order to name a query file that will represent what is the query for ,but in the same time to be easy to read, the file name will be constructed from five parts. These are operation, rdf type, class name, property name and object property name. All names are only the local segment of the Qname (compressed URI) provided and the rdf types are instance, property and reified (reified property).

File name examples:

Structure → `operation_rdfType_className`

File name: added_instance_collection

Structure → `operation_rdfType_className_propertyName`

File name: changed_property_concept_broader

10.3 Structure

In this project the SPARQL query is constructed from five parts that will be explained below.

Prefixes section

Prefixes section are declared in here before the select statement of the query. It can have as many possible prefixes and values as the query will use only the ones that it needs.

Query variables

A SPARQL query file could sometimes be long and hard to read. To improve readability and minimize use of hidden variables, the query parameters should express in some manner what is the query used for and also to have the same values in the entire query. For easy referencing a change from the SPARQL query parameters, a good option is to add a prefix to the parameters that will have the value changed in the diffing context. To avoid pollutions of variables names the convention will add prefix in front of the variables that is changing in the diffing context by using the query. This can only be old or new (i.e ?oldInstance ?newInstance).

Version history graph block

This block will remain unchanged for all diffing queries as it's defining the graphs that are used later in the query. As a mention there is only one part that can change here and that is the value injection for the query which will be present in the next section. The logic of the query is based on this four graphs that are built here. We can see below that we are going to have access to newVersionGraph, oldVersionGraph, insertionsGraph and deletionsGraph, so we can filter our data.

```
GRAPH ?versionHistoryGraph {

  # parameters
  VALUES ( ?versionHistoryGraph ?oldVersion ?newVersion ?class) {
      ( undef
        undef
        undef
        skos:Concept
      )
  }
  # get the current and the previous version as default versions
  ?versionset dsv:currentVersionRecord/xhv:prev/dc:identifier ?
     previousVersion .
  ?versionset dsv:currentVersionRecord/dc:identifier ?latestVersion
      .
  # select the versions to actually use
```

```
  BIND(coalesce(?oldVersion, ?previousVersion) AS ?
     oldVersionSelected)
  BIND(coalesce(?newVersion, ?latestVersion) AS ?newVersionSelected
     )
  # get the delta and via that the relevant graphs
  ?delta a sh:SchemeDelta ;
    sh:deltaFrom/dc:identifier ?oldVersionSelected ;
    sh:deltaTo/dc:identifier ?newVersionSelected ;
    sh:deltaFrom/sh:usingNamedGraph/sd:name ?oldVersionGraph ;
    sh:deltaTo/sh:usingNamedGraph/sd:name ?newVersionGraph .
  ?insertions a sh:SchemeDeltaInsertions ;
    dct:isPartOf ?delta ;
    sh:usingNamedGraph/sd:name ?insertionsGraph .
  ?deletions a sh:SchemeDeltaDeletions ;
    dct:isPartOf ?delta ;
    sh:usingNamedGraph/sd:name ?deletionsGraph .
}
```

Value injection

As mentioned in the previous section there is a value injection block where we can assign values to variables that are going to be used in the query logic.

```
# parameters
VALUES ( ?versionHistoryGraph ?oldVersion ?newVersion ?class) {
    ( undef
      undef
      undef
      skos:Concept
    )
}
```

Query logic

This part of the query is filtering the data by looking for triples in the graphs made available in the version history graph block. The graphs are made available through the delta generated (see Versions and Deltas as Named Graphs) by the diffing process, and they are as follows:

- oldVersionGraph - contains triples existing in the old version file
- newVersionGraph - contains triples existing in the new version file
- insertionsGraph - contains added triples to the old version file

- deletionsGraph - contains triples deleted from the old version file

As an example, will want to look for new instances that are of a certain class and to achieve this will want to look into the insertions graph.

```
GRAPH ?insertionsGraph {
  ?instance a ?class .

  optional {
    ?instance skos:prefLabel ?prefLabelEn .
    FILTER (lang(?prefLabelEn) = "en")
  }
}
```

The query logic can continue to filter the results by verifying existence of triples in other available graphs. This can be done by using another graph block as showed below.

```
# ... and the instance must not exist in the old version
FILTER NOT EXISTS {
  GRAPH ?oldVersionGraph {
    ?instance ?p [] .
  }
}
```

10.4 Example of diffing query

Building a query to get all added skos:altLabel property per instance of a certain class.

Prefixes section

```
# basic namespaces
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
.
.
.
# versioning namespaces
PREFIX dsv: <http://purl.org/iso25964/DataSet/Versioning#>
PREFIX sd: <http://www.w3.org/ns/sparql-service-description#>
PREFIX sh: <http://purl.org/skos-history/>
PREFIX xhv: <http://www.w3.org/1999/xhtml/vocab#>
```

Query variables

For the results and the query itself to be easy to read we need to choose good variable names and make sure the variables will bring the expected result format. In this case we want to see instance URI, instance label, property and value of the property.

```
SELECT DISTINCT ?instance ?prefLabel ?property ?value
WHERE {
```

Version history graph block and value injection

```
GRAPH ?versionHistoryGraph {
  # defining values for our variables that are we going to use in
      the query (instance class and property)
  VALUES ( ?versionHistoryGraph ?oldVersion ?newVersion ?class ?
      property) {
    ( undef
      undef
      undef
      skos:Concept
      skos:altLabel
    )
  }
  # get the current and the previous version as default versions
  ?versionset dsv:currentVersionRecord/xhv:prev/dc:identifier ?
      previousVersion .
  ?versionset dsv:currentVersionRecord/dc:identifier ?latestVersion
      .
  # select the versions to actually use
  BIND(coalesce(?oldVersion, ?previousVersion) AS ?
      oldVersionSelected)
  BIND(coalesce(?newVersion, ?latestVersion) AS ?newVersionSelected
      )
  # get the delta and via that the relevant graphs
  ?delta a sh:SchemeDelta ;
    sh:deltaFrom/dc:identifier ?oldVersionSelected ;
    sh:deltaTo/dc:identifier ?newVersionSelected ;
    sh:deltaFrom/sh:usingNamedGraph/sd:name ?oldVersionGraph ;
    sh:deltaTo/sh:usingNamedGraph/sd:name ?newVersionGraph ;
    dct:hasPart ?insertions ;
    dct:hasPart ?deletions .
  ?deletions a sh:SchemeDeltaDeletions ;
    sh:usingNamedGraph/sd:name ?deletionsGraph .
  ?insertions a sh:SchemeDeltaInsertions ;
    sh:usingNamedGraph/sd:name ?insertionsGraph .
}
```

Query logic

In this part we need to filter the results to get only the instances that had the skos:altLabel property added. As a starting point we will look in the insertions graph to get all inserted skos:altLabel properties for all instances. For this to be a true addition and not a change or a movement operation we need to make sure that the property was not attached to some other instance before and to do this will look into the deletions graph and old version graph. After filtering the result we can get all the values that are needed from the new version graph.

```
# get inserted properties for instances
GRAPH ?insertionsGraph {
    ?instance ?property [] .
  }
  # ... which were not attached to some (other) instance before
  FILTER NOT EXISTS {
    GRAPH ?deletionsGraph {
      ?instance ?property [] .
    }
  }
  FILTER NOT EXISTS {
    GRAPH ?oldVersionGraph {
      [] ?property ?value .
    }
  }
  # get instances with those property values
  GRAPH ?newVersionGraph {
    ?instance a ?class .
    ?instance ?property ?value .

    optional {
      ?instance skos:prefLabel ?prefLabel .
      FILTER (lang(?prefLabelEn) = "en")
    }
  }
}
```

Appendices

References

[1] E. Costetchi. Asset publication lifecycle architecture. Recommendation, Publications Office of the European Union, September 2020.

[2] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[3] Solomon Hykes. Docker, 2013. URL `http://www.docker.com`.

[4] The Apache Software Foundation. Apache Jena Fuseki, 2011. URL `https://jena.apache.org/documentation/fuseki2/`.