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 asset publishing workflow services

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 1.3.8: Installation guide for the asset publishing workflow services
Corporate Author	Publications Office of the European Union
Author	Eugeniu Costetchi
Reviewers	Denis Dechandon and Willem Van Gemert
Contractor	Infeurope S.A.
Framework contract	10688/35368
Work package	WP 1.3.8
Delivery date	06 November 2020
Suggested readers	technical staff, system administrators, enterprise architects, software developers

© European Union, 2020

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 [2] 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 [7], 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. RDF validator
3. RDF fingerprinter
4. Fuseki triplestore [11]
5. LinkedPipes ETL [4, 3]
6. Jenkins automation server [5]
7. Camunda BPMN platform [1]
8. NginX HTTP server [9]

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 redhat. (reason: project requirement)

Also present:

Infrastructure and deployment configuration rely on the *Docker technology* [7, 10]. 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

RedHat.

The inventory of pre-configured ports is provided in Table 1.

Service name	HTTP port UI	HTTP port API	FTP port	Mounted volume
RDF differ	8030	4030		
RDF differ dedicated Fuseki		3030		rdf-differ-fuseki
RDF validator	8010	4010		
RDF fingerprinter	8020	4020		
RDF fingerprinter dedicated Fuseki		3020		rdf-fingerprinter-fuseki
LinkedPipes ETL - storage		8063		linkedpipes-logs, linkedpipes-data-storage, linkedpipes-configuration
LinkedPipes ETL - executor		8065		linkedpipes-logs, linkedpipes-data-execution, linkedpipes-configuration
LinkedPipes ETL - monitor		8061	2221, 2222, 2225	linkedpipes-logs, linkedpipes-data-execution, linkedpipes-configuration

Table 1 continued from previous page

LinkedPipes ETL - frontend	8060		linkedpipes-logs, linkedpipes-configuration
LinkedPipes ETL - dedicated Fuseki		3060	linkedpipes-fuseki
Jenkins	8080	50000	jenkins-home
Camunda BPMN engine	8040		rdf-camunda

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

Change directory into the project folder to be able to use the Makefile commands to start and stop services will be available.

In order to run the services it is necessary to have the project dependencies installed.

To do so, run the following commands:

```
make install-os-dependencies
```

To start the services using Makefile

```
make start-services
```

To stop the services using Makefile

```
make stop-services
```

To start services without Makefile first prepare the volume with LinkedPipes ETL configurations file like this

```
docker rm temp | true
docker volume rm linkedpipes-configuration | true
docker volume create linkedpipes-configuration
docker container create --name temp -v linkedpipes-configuration:/
    data busybox
docker cp ./docker/linkedpipes-etl/configuration/configuration.
    properties temp:/data
docker rm temp
```

then start the services

```
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
```

6.1 Set custom templates

RDF Differ, RDF Validator and RDF Fingerprinter support custom templates. The custom templates are stored in Docker volumes, and are explicitly marked to be externally created. Therefore, before running the service for the first time, ensure the volumes exist by running the following make command:

```
make build-template-volumes
```

Provided that the custom template is available on the host system in a *location to template* you can set it by using the following makefile targets, which are prepared to facilitate this operation.

```
make location=<location to template> differ-set-report-template
make location=<location to template> validator-set-report-template
make location=<location to template> fingerprinter-set-report-
    template
```

The detailed explanation on how to configure them is provided in the Configuration section for each of these services (See Section 7.1, 7.3, 7.5).

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.

Custom template configuration

The **default diff report** template resides in
`/usr/src/app/resources/templates/diff_report`.

The custom template functionality is implemented using **docker's volumes** mechanism. This implementation has been chosen as it requires no code modifications from the end-user's side.

An externally defined volume `rdf-differ-template` which will contain the externally defined template (i.e. the custom template), which in turn is coupled with the `rdf-differ-api` docker container to use when generating the reports. The coupling of the volume to the service container is done with the following statement, which is included in the default docker compose configuration.

```
volumes:
- rdf-differ-template:${RDF_DIFFER_TEMPLATE_LOCATION}
```

The lines above map the custom template that has been copied to the docker volume with the internal location of the container which has been defined in the `.env` file.

`RDF_DIFFER_TEMPLATE_LOCATION` is an environment variable used in the internal implementation of the `rdf-differ` service. Because it is internal it was not mentioned in Table 2.

To configure your own template you can copy the default report template and adjust it to your needs or design a new one from scratch. The templates are written in *Jinja2* templating language [8]. The data source access is facilitated through the *eds4jinja2* library [6]. 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].

Use the custom template

After you have your custom template, run the `make` command, indicating the location of your template through the `location` variable.

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

[2] `https://github.com/meaningfy-ws/rdf-differ/tree/master/resources/templates/diff_report`

```
make location=<location to template> differ-set-report-template
```

NOTE: Make sure that the location specified ends with a trailing slash `/`, otherwise the command will not work propery and the templates will not be copied to the docker volume.

Example:

```
make location=~/template/location/ differ-set-report-template
```

After this, restart the `rdf-differ-api` container for the effects to take place.

7.2 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 3: RDF differ dedicated triple store configurations

7.3 RDF fingerprinter

RDF fingerprinter application exposes an API and an UI. It is based on executing SPARQL queries on given data and therefore also needs a dedicated triple store service.

Description	Value	Associated variable
Service UI domain	http://rdf-fingerprinter-ui	RDF_FINGERPRINTER_UI_LOCATION
Service UI port	8020	RDF_FINGERPRINTER_UI_PORT
Service API domain	http://rdf-fingerprinter-api	RDF_FINGERPRINTER_API_LOCATION
Service API port	4020	RDF_FINGERPRINTER_API_PORT

Table 4: RDF fingerprinter configuration

Please note that the URL is only available inside the same 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.

Custom Template Configuration

The **default fingerprinter report** template resides in the python fingerprinter package: `rdf-fingerprinter`.

The custom template functionality is implemented using **docker's volumes** mechanism. This implementation has been chosen as it requires no code modifications from the end-user's side.

An externally defined volume `rdf-fingerprinter-template` which will contain the externally defined template (aka: the custom template), which in turn is coupled with the `rdf-fingerprinter-api` docker container to use when generating the reports. The coupling of the volume to the service container is done with the following statement, which is included in the default docker compose configuration.

```
volumes:
- rdf-fingerprinter-template:${RDF_FINGERPRINTER_TEMPLATE_LOCATION}
```

`RDF_FINGERPRINTER_TEMPLATE_LOCATION` is an environment variable used in the internal implementation of the `rdf-fingerprinter` service.

The lines above map the custom template that has been copied to the docker volume with the internal location of the container which has been defined in the `.env` file.

To configure your own template you can copy the default report template and adjust it to your needs or design a new one from scratch. The templates are written in *Jinja2* templating language [8]. The data source access is facilitated through the *eds4jinja2* library [6]. If you are familiar with Jinja2 language a short introduction to how to use eds4jinja2 is available on the documentation page[3]. Also the default template can be seen as an example accessible in the repository[4].

Use the custom template

After you have your custom template, run the `make` command, indicating the location of your template through the `location` variable.

```
make location=<location to template> fingerprinter-set-report-
    template
```

NOTE: Make sure that the location specified ends with a trailing slash `/`, otherwise the command will not work propery and the templates will not be copied to the docker volume.

Example:

```
make location=~/template/location/ fingerprinter-set-report-
    template
```

After this, restart the `rdf-fingerprinter-api` container for the effects to take place.

7.4 RDF fingerprinter dedicated triple store

Fuseki triple store is used as the supporting triple store for this service. The available configurations for the Fuseki are described below.

Description	Value	Associated variable
Admin password	admin	RDF_DIFFER_FUSEKI_ADMIN_PASSWORD

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

[4] `https://github.com/meaningfy-ws/rdf-fingerprinter/tree/master/fingerprint_report_templates/fingerprint_report`

Table 5 continued from previous page

User name	admin	RDF_FINGERPRINTER_FUSEKI_USERNAME
Password	admin	RDF_FINGERPRINTER_FUSEKI_PASSWORD
Fuseki data folder	./data	RDF_FINGERPRINTER_FUSEKI_DATA_FOLDER
External port	3020	RDF_FINGERPRINTER_FUSEKI_PORT
Additional JVM arguments	-Xmx2g	RDF_DIFFER_FUSEKI_JVM_ARGS
Service URL	http://rdf-differ-fuseki	RDF_DIFFER_FUSEKI_LOCATION

Table 5: RDF differ dedicated triple store configurations

7.5 RDF validator

RDF validator application exposes an API and an UI and does not depend on any additional services as everything is encapsulated into the Docker image. The configuration options are summarised below.

Description	Value	Associated variable
Service UI port	8010	VALIDATOR_UI_PORT
URL	http://rdf-validarot-ui:8010	RDF_VALIDATOR_UI_URL
Service API port	4010	VALIDATOR_API_PORT

Table 6: RDF validator configurations

Note, when validating SPARQL endpoints, the fully qualified domain name of the machine must be specified. As a consequence, "localhost" domain will not work as expected.

Custom Template Configuration

The **default validator report** template resides in `/usr/src/app/resources/templates/validator_report`.

The custom template functionality is implemented using **docker's volumes** mechanism. This implementation has been chosen as it requires no code modifications from the end-user's side.

An externally defined volume `rdf-validator-template` which will contain the externally defined template (aka: the custom template), which in turn is coupled with the `rdf-validator-api` docker container to use when generating the reports. The coupling of the volume to the service container is done with the following statement, which is included in the default docker compose configuration.

```
volumes:
- rdf-validator-template:${RDF_VALIDATOR_TEMPLATE_LOCATION}
```

`RDF_VALIDATOR_TEMPLATE_LOCATION` is an environment variable used in the internal implementation of the `rdf-validator` service.

The lines above map the custom template that has been copied to the docker volume with the internal location of the container which has been defined in the `.env` file.

To configure your own template you can copy the default report template and adjust it to your needs or design a new one from scratch. The templates are written in *Jinja2* templating language [8]. The data source access is facilitated through the *eds4jinja2* library [6]. If you are familiar with Jinja2 language a short introduction to how to use eds4jinja2 is available on the documentation page[5]. Also the default template can be seen as an example accessible in the repository[6].

Use the custom template

After you have your custom template, run the `make` command, indicating the location of your template through the `location` variable.

```
make location=<location to template> validator-set-report-template
```

NOTE: Make sure that the location specified ends with a trailing slash `/`, otherwise the command will not work propery and the templates will not be copied to the docker volume.

Example:

```
make location=~/template/location/ validator-set-report-template
```

After this, restart the `rdf-validator-api` container for the effects to take place.

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

[6] `https://github.com/meaningfy-ws/rdf-validator-ws/tree/master/resources/templates/validator_report`

7.6 Nginx server

Nginx is a web server and in this context it serves on the port 80 (default HTTP) a splash page. However it can be configured in the future to operate as a reverse proxy as it may be necessary in the deployed environment. No configurations are foreseen for this service at the moment.

7.7 Jenkins automation server

Jenkins automation server can be used to orchestrate some workflows especially those that may be triggered by operations on the SVN common repository. Only the port configurations are foreseen at the moment through environment variables. Additional ones can be done by following the official Jenkins installation manual.

Description	Value	Associated variable
Service UI portUI	8080	JENKINS_UI_PORT
Agent port	50000	JENKINS_AGENTS_PORT

Table 7: Jenkins automation server configurations

7.8 LinkedPipes ETL services

LinkedPipes ETL is deployed as a set of four dockerised services: storage, executor, executor monitor and the user interface. Additionally a dedicated triple store is also considered and described in the next section.

LinekdPipes ETL services are configured with (a) a set of environment variables to control the Docker containers and (b) a special configurations file (*configurations.properties*), which is used natively by the LinkedPipes ETL components (running inside the container). This configurations file mirrors the established environment variables enumerated below.

Description	Value	Associated variable
Storage service port	8063	LP_ETL_STORAGE_PORT
Executor service port	8065	LP_ETL_EXECUTOR_PORT
Executor monitor service port	8061	LP_ETL_MONITOR_PORT
Service UI port	8060	LP_ETL_PORT

Table 8 continued from previous page		
Service domain	http://localhost:8060	LP_ETL_DOMAIN

Table 8: LinkedPipes ETL services configurations

Note that it is important to change the LP_ETL_DOMAIN each time the deployment environment changes. If it runs locally please set the value to be *http://localhost:8060*, otherwise if it runs on a dedicated domain, the variable value must be set accordingly. If this is not done, the services will run but will not be able to load any pipelines because of a mismatch between the domain in the pipeline URI (created by using the variable value) and the domain of the service host.

The special *configuration.properties* file[7], as mentioned above must be in synch with the port numbers of the environment variables. The meaning of these variables is explained on the LinkedPipes ETL website. They have been preconfigured, so that no changes are necessary there but of course they can be adjusted if needed. The native configuration file is available next to the *docker-compose.yml* in the subfolder *linkedpipes-etl/configuration*. The content of this file with descriptions of parameters as provided by the LinkedPipes ETL authors is available in the Section 8.

7.9 LinkedPipes ETL dedicated triple store

LinkedPipes ETL dedicated triple store is provided as an operational space to support the ETL workflows. The configurations are minimal as indicated below.

Description	Value	Associated variable
Admin password	admin	LP_ETL_FUSEKI_ADMIN_PASSWORD
Additional arguments passed to JVM	-Xmx2g	LP_ETL_FUSEKI_JVM_ARGS
Fuseki port	3060	LP_ETL_FUSEKI_PORT

Table 9: LinkedPipes ETL dedicated triple store configurations

[7] `https://github.com/meaningfy-ws/mdr-workflow/blob/master/docker/linkedpipes-etl/configuration/configuration.properties`

7.10 Camunda BPMN engine

Camunda BPMN engine is deployed as a stand alone service. A minimal set of configurations are provided here and more advanced ones shall be performed following the official installation manual.

Description	Value	Associated variable
Service UI port	8080	CAMUNDA_UI_PORT

Table 10: Camnunda BPMN service configurations

Appendices

8 LinkedPipes ETL adapted configurations file

```
#
# LinkedPipes ETL Configuration file adapted for MDR Workflow
#

###############
#  Executor  #
###############

# Port used by executor REST API.
executor.webserver.port = 8065

# URL of the executor REST API to be used. Must NOT end with '/'.
# This can be used for setups where executor and executor-monitor
# run on one machine and the frontend on another. If all components
# run on the same machine, this is http://localhost:8065 or the
# port from executor.webserver.port
executor.webserver.uri = http://linkedpipes-etl-executor:8065

# Path to debug data. Debug data can be large.
# Linux ex: /data/lp/etl/working
executor.execution.working_directory = /data/lp/etl/executor

# Directory used for logs.
# Linux ex: /data/lp/etl/log
executor.log.directory = /data/lp/etl/logs

# TRACE, DEBUG, INFO, WARN, ERROR
executor.log.core.level = INFO

# Path to utilized libraries. This is usually the deploy/osgi
# folder where you cloned the repository.
# Linux ex: /opt/lp/etl/deploy/osgi
executor.osgi.lib.directory = /opt/lp/etl/osgi

# Path to OSGI working directory, used bundles are stored here.
executor.osgi.working.directory = /data/lp/etl/executor/felix

# List of regexp pattern. Every component has an IRI if the IRI
    match any of
# the listed patterns then attempt to execute such component cause
# pipeline to fail.
# The default value ban components that are working with local
    resources.
```

```
# executor.banned_jar_iri_patterns = ".*e-filesFromLocal.*",".*l-
    filesToLocal.*",".*x-deleteDirectory.*"

######################
#  Executor-monitor  #
######################

# Port used by executor-monitor REST API.
executor-monitor.webserver.port = 8061

# URL of the executor-monitor REST API to be used.
# Must NOT end with '/'.
executor-monitor.webserver.uri = http://linkedpipes-etl-executor-
    monitor:8061

# Directory used for logs.
# Linux ex: /data/lp/etl/log
executor-monitor.log.directory = /data/lp/etl/logs

# TRACE, DEBUG, INFO, WARN, ERROR
executor-monitor.log.core.level = INFO

# FTP Port for executor-monitor.
# The FTP server is used to browse debug content of an execution.
executor-monitor.ftp.command_port = 2221
executor-monitor.ftp.data_ports_interval.start = 2222
executor-monitor.ftp.data_ports_interval.end = 2225

# External URL that should be used to access debug FTP server from
    outside.
# This must point to the FTP server instance command port.
# Must NOT end on '/'.
executor-monitor.ftp.uri = ftp://linkedpipes-etl-executor-monitor
    :2221

# Optional property, can be used to send notification to slack
# about pipeline successfully finished executions.
# executor-monitor.slack_finished_executions_webhook =

# Optional property, can be used to send notification to slack.
# Include executions related error/cancelled messages.
# executor-monitor.slack_error_webhook =

# Optional property, can be used to provide alternative access to
    debug files.
# If provided the the URL must resolve to the
# 'executor.execution.working_directory' and any sub-path to
# appropriate file in this directory.
```

```
# This property allow to use custom service to provide access to
    debug data.
#executor-monitor.public_working_data_url_prefix = http://localhost

###############
#  Storage    #
###############

# URL of the storage REST API to be used. Must NOT end with '/'.
# If all components run on the same machine, this is
# http://localhost:8063 or the port from storage.port
storage.uri = http://linkedpipes-etl-storage:8063

# Port used by storage REST API.
storage.port = 8063

# Linux ex: /opt/lp/etl/deploy/jars
storage.jars.directory = /opt/lp/etl/components

# Directory used by the storage.
# Linux ex: /data/lp/etl/storage
storage.directory = /data/lp/etl/storage

# Prefix used to create URI of templates and pipelines, must be
    dereferencable.
# Must NOT end with '/'.
domain.uri = http://linkedpipes-etl-frontend:8060

# Directory used for logs.
# Linux ex: /data/lp/etl/log
storage.log.directory = /data/lp/etl/logs

# TRACE, DEBUG, INFO, WARN, ERROR
storage.log.core.level = INFO

###############
#  Frontend   #
###############

# Port for web server.
frontend.webserver.port = 8060

# Frontend Title
frontend.instance-label = LinkedPipes ETL
```

References

[1] Camunda. Camunda BPM, 2013. URL https://camunda.com/.

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

[3] J. Klímek and P. Škoda. Linkedpipes etl in use: practical publication and consumption of linked data. In *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services*, pages 441–445, 2017.

[4] J. Klímek, P. Škoda, and M. Nečaskỳ. Linkedpipes etl: Evolved linked data preparation. In *European Semantic Web Conference*, pages 95–100. Springer, 2016.

[5] Kohsuke Kawaguchi. Jenkins, 2011. URL https://www.jenkins.io/.

[6] Meaningfy.ws. Embedded Datasource Specification in Jinja2 templates, 2020. URL https://eds4jinja2.readthedocs.io/en/latest/.

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

[8] Pallets. Jinja 2 templating language, 2007. URL https://jinja.palletsprojects.com/en/2.11.x/.

[9] W. Reese. Nginx: the high-performance web server and reverse proxy. *Linux Journal*, 2008(173):2, 2008.

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

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