Publications Office of the European Union

infeurope

Initiative for Modeling the Legal Analysis Methodology

# Technical manual for the LAM content generation service

# Disclaimer

The views expressed in this report are purely those of the Author(s) and may not, in any circumstances, be interpreted as stating an official position of the European Commission. The European Commission 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 Commission. All care has been taken by the author to ensure that s/he has obtained, where necessary, permission to use any parts of manuscript s including illustrations, maps, and graphs, on which intellectual property rights already exist from the titular holder(s) of such rights or from her/his or their legal representative.

## Document metadata

| | |
|---|---|
| **Reference** | WP 4.1 & WP 4.2: Technical manual for the LAM content generation service |
| **Corporate Author** | Publications Office of the European Union |
| **Author** | Eugeniu Costetchi |
| **Reviewers** | Juraj Kuba, Miltiadis Charitos, Marcin Baryn |
| **Contractor** | Infeurope |
| **Framework contract** | FC No 10688/ OF No 36269 |
| **Work package** | WP 4.1 & WP 4.2 |
| **Delivery date** | 19 Feb 2021 |
| **Suggested readers** | system administrators, software developers, future users |

# Abstract

This document provides technical guidance on how to install and configure LAM Validator and LAM content generator, two micro-services developed in the context of the initiative for modelling the Legal Analysis Methodology (LAM).

The LAM validator is used to perform validation of RDF content after it is exported from VocBench 3 platform. The validation procedure is based on SHACL data shape checking and is done using LAM-SKOS-AP application profile.

The LAM content generator is used to generate, from valid RDF files, the HTML representation, PDF representation and index files in JSON representation.

# Contents

# 1 Introduction

This document provides technical guidance on how to install and configure LAM Validator and LAM content generator, two micro-services developed in the context of the initiative for modelling the Legal Analysis Methodology (LAM).

The LAM validator is used to perform validation of RDF content after it is exported from VocBench 3 platform. The validation procedure is based on SHACL data shape checking and is done using LAM-SKOS-AP application profile.

The LAM content generator is used to generate, from valid RDF files, the HTML representation, PDF representation and index files in JSON representation.

The employed infrastructure technology for the micro-service management is based on Docker introduced 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. LAM RDF validator
2. LAM content generator

# 3 Target audience

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

- System administrators
- Developers in charge
- Technical users

# 4 Technology background

Infrastructure and deployment configuration rely on the *Docker technology* [1, 2]. 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 5.4.x 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 | FTP port | Mounted volume |
|---|---|---|---|---|
| RDF validator | 8010 | 4010 | | |

Table 1: Port usage inventory

The minimal hardware requirements are as follows

1. CPU: Dual core 3Ghz
2. RAM: 8Gb
3. SDD system: 2Gb
4. SDD data: 8Gb

# 6 Installation

In order to run the services it is necessary to have Docker [2] service and docker-compose tool installed. To install them follow the instructions provided on the official websites

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
```

Please note that the *docker.io* package si installed rather than the *docker-ce* one.

Next, to launch the services, clone the Git repository containing the *docker-compose.yml*, *.env* file and the *Makefile*.

```
git clone https://github.com/meaningfy-ws/lam-workflow.git
cd lam-workflow
```

You may choose to adjust these files as necessary on your system. Then change directory into the *lam-workflow* folder and Makefile commands to start and stop services will be available. To start services run

```
make start-services
```

To stop the services run

```
make stop-services
```

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

To start the services using Makefile

```
make location=</your-custom/shapes/location> validator-set-shacl-
    shapes
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 rdf-validator-shacl-shapes | true
docker volume create rdf-validator-shacl-shapes
docker container create --name temp -v rdf-validator-shacl-shapes:/
    data busybox
docker cp <your-custom/shapes/location>. 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
```

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

# 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 Validator API Docker container, which is defined, in the `docker-compose.yml` file as it follows:

```
lam-validator-api:
        container_name: lam-validator-api
        image: meaningfy/lam-validator-api:latest
        ports:
                - ${RDF_VALIDATOR_API_PORT}:${
                   RDF_VALIDATOR_API_PORT}
        env_file: .env
        restart: always
        networks:
                - mydefault
```

The variable used in the definition of this service is just one, `RDF_VALIDATOR_API_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_VALIDATOR_API_PORT=10001`. Change the value of the port, rebuild the micro-services and RDF Differ will no longer be listening on `10001`, but on the new port that you specified.

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

## 7.1 LAM Validator

LAM 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 |
|---|---|---|
| Validator Service UI port | 10002 | RDF_VALIDATOR_UI_PORT |
| Validator Service UI location | http://lam-validator-ui | RDF_VALIDATOR_UI_LOCATION |
| Validator Service API port | 10001 | RDF_VALIDATOR_API_PORT |
| Validator Service API location | http://lam-validator-api | RDF_VALIDATOR_API_LOCATION |

Table 2: LAM Validator configuration

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.

### Configure SHACL Shapes Files

The docker image for the LAM validation API service pulled from **docker hub** already contains the required SHACL shape files, as defined in this **repository**.

If you want to change the SHACL shapes files, you can use the following customization implemented using **docker's volumes** mechanism. This implementation has been chosen as it requires no in service code modifications from the end-user's side.

An externally defined volume `rdf-validator-shacl-shapes` which will contain the custom files is coupled with the `rdf-validator-api` docker container to use when validating. The coupling of the volume to the service container is done with the following statement, which is not included in the default docker compose configuration.

```
volumes:
- rdf-validator-shacl-shapes:${RDF_VALIDATOR_SHACL_SHAPES_LOCATION}
```

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

You have to copy these 2 lines in the `lam-validator-api` container definition in the `docker-compose.yml` file after the `image: meaningfy/lam-validator-api:latest` line.

Additionally, the externally defined volume has to be specified in the `docker-compose.yml` file:

```
volumes:
 rdf-validator-shacl-shapes:
  external: true
```

To make the custom shapes available to the container create the volume and run the `make` commands, indicating the location of your shapes through the `location` variable.

```
make build-volumes
make location=<location to shapes> set-shacl-shapes
```

**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=/shapes/location/ set-shacl-shapes
```

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

## 7.2 LAM Generation Service

LAM Generation Service application exposes an API and an UI for generating the LAM reports (in HTML and PDF formats) and the document index files. It depends on a dedicated triple store which will contain the LAM dataset indicated by the `LAM_FUSEKI_QUERY_URL` described in table 4 environment variable.

The configuration options are summarised below.

| Description | Value | Associated variable |
|---|---|---|
| LAM Generation Service UI port | 8050 | `LAM_UI_PORT` |
| LAM Generation Service UI location | http://lam-generation-service-ui | `LAM_UI_LOCATION` |
| LAM Generation Service API port | 4050 | `LAM_API_PORT` |

**Table 3 continued from previous page**

| | | |
|---|---|---|
| LAM Generation Service API location | http://lam-generation-service-api | LAM_API_LOCATION |

Table 3: LAM Generation Service configuration

## 7.3 LAM dedicated triple store

LAM Generation Services depends on a Fuseki triple store and query the data required for the LAM reports and index files.

The available configurations are described below.

| Description | Value | Associated variable |
|---|---|---|
| Admin account password | admin | LAM_FUSEKI_ADMIN_PASSWORD |
| User name | admin | LAM_FUSEKI_USERNAME |
| Password | admin | LAM_FUSEKI_PASSWORD |
| Folder where Fuseki stores data | ./data/diff | LAM_FUSEKI_DATA_FOLDER |
| Additional arguments passed to JVM | -Xmx2g | LAM_FUSEKI_JVM_ARGS |
| URL | http://rdf-differ-fuseki | LAM_FUSEKI_LOCATION |
| LAM Fuseki port | 3030 | LAM_FUSEKI_PORT |
| LAM Fuseki External port | 3010 | LAM_FUSEKI_EXTERNAL_PORT |
| LAM Fuseki location | http://lam-fuseki | LAM_FUSEKI_LOCATION |
| Fuseki LAM dataset location | /lam/query | LAM_FUSEKI_QUERY_URL |

Table 4: LAM Generation Services dedicated triple store configuration

# Appendices

# References

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

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