H2020-EINFRA-2017

EINFRA-21-2017 - Platform-driven e-infrastructure innovation

DARE [777413] "Delivering Agile Research Excellence on European e-Infrastructures"

D3.4 Data Lineage Services II

Project Reference No	777413 — DARE — H2020-EINFRA-2017 / EINFRA-21-2017
Deliverable	D3.4 Data Lineage Services II
Work package	WP3: Large-scale Lineage and Process Management
Tasks involved	T3.2, T3.3
Type	DEM: Demonstrator, pilot, prototype
Dissemination Level	PU = Public
Due Date	31/12/2020
Submission Date	30/12/2020
Status	Draft
Editor(s)	Alessandro Spinuso
Contributor(s)	
Reviewer(s)	Horst Schwichtenberg (SCAI)
Document description	Overview of the third intermediate release of the Lineage Services.

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
1	12/03/2020	Outline and inclusion of Lineage API	Alessandro Spinuso
1	25/09/2020	Refinement of outline and update content	Alessandro Spinuso
1	23/09/2020	Conclusions and Future Work	Alessandro Spinuso
1	15/12/2020	Applied Revision Comments	Alessandro Spinuso

Executive Summary

This document describes the work conducted for the last release of the DARE lineage service API. In Section 2 we address the improvements of its architecture and deployment, which includes new components to accommodate the AAI (Authentication Authorisation Infrastructure) of the DARE platform and resiliency queues. Section 4 instead illustrates the new search functionalities, followed by the preliminary support for the acquisition of lineage produced by CWL workflows (Section 5). Finally the API has been equipped with a new test framework, which is integrated within the CI/CD pipeline of the S-ProvFlow system (Section 6).

Table of Contents

List of Terms and Abbreviations

Abbreviation	Definition
PROV	W3C Standard for Provenance Representation
S-PROV	PROV extension for lineage representation of streaming operators.
CWLProv	Provenance Information produced by CWL Workflows
CI/CD	Continuous Integration/Deployment
AAI	Authentication Authorisation Infrastructure
CWL	Common Workflow Language

1 Introduction

We present the updates of the S-ProvFlow Lineage services. These have been conducted to achieve a full integration with the DARE platform, especially concerning the identification and authentication of the user and its resiliency to temporary downtime that might occur to its persistence components. The developments also improved the usability of the API, thanks to the enhanced metadata search capabilities, and its overall quality, which is now provided with CI/CD pipelines and a testing framework.

The experimental adoption of CWL for the implementation of new use cases triggered the investigation of extending the API with capabilities for the acquisition of lineage expressed in CWLProv. This produced a preliminary implementation, which also triggered considerations on the coverage and usability challenges associated with the support of this format.

2 Updated Architecture

The S-ProvFlow system[1] integrated in DARE combines a set of components that support acquisition and exploration of lineage and provenance data produced by the workflows. It includes a database, a web service layer and two complementary interactive tools. The whole system is delivered as a composition of different Docker containers following a microservices deployment approach in DARE. Beyond the modularity and decoupling, which is typical of microservices architectures, this choice facilitates ways to make it accessible to research-developers and administrators who want to explore and investigate issues, or keep the system updated. In Figure 1 we show how the microservices-based deployment is currently setup within the DARE platform.

2.1 Messages Queues

Message and fail-over queues (S-ProvFlow Queue) have been implemented to detach the workflow execution from the direct access to the provenance database services. This has the advantage of delegating to the queue those mechanisms that can recover from a temporary unavailability of the provenance API (S-ProvFlow API), preventing message loss. Moreover, this architecture concentrates in the queue the overhead of authenticating and storing the provenance messages into the database, with less impact on the workflow's execution, especially on managing the authentication and failovers.

2.2 AAI

Authentication is integrated via the adoption of delegation tokens (OAuth2). These are sent from the workflow to the queue, which thereby uses them to authenticate to the API via a dedicated Keycloak Gate-keeper. Users are identified anonymously and uniquely by using a combination of the account identifier (*sub*) and the identifier for the identity provider who issued (*iss*) the credentials. This approach also supports with the GDPR regulations in terms of privacy and anonymity of information associated with the users and their activities across systems. The interactive monitoring tools also make use of authentication mechanisms. We will describe this in more detail in D-3.8.

[1] https://gitlab.com/project-dare/s-ProvFlow/-/tree/1.2.7

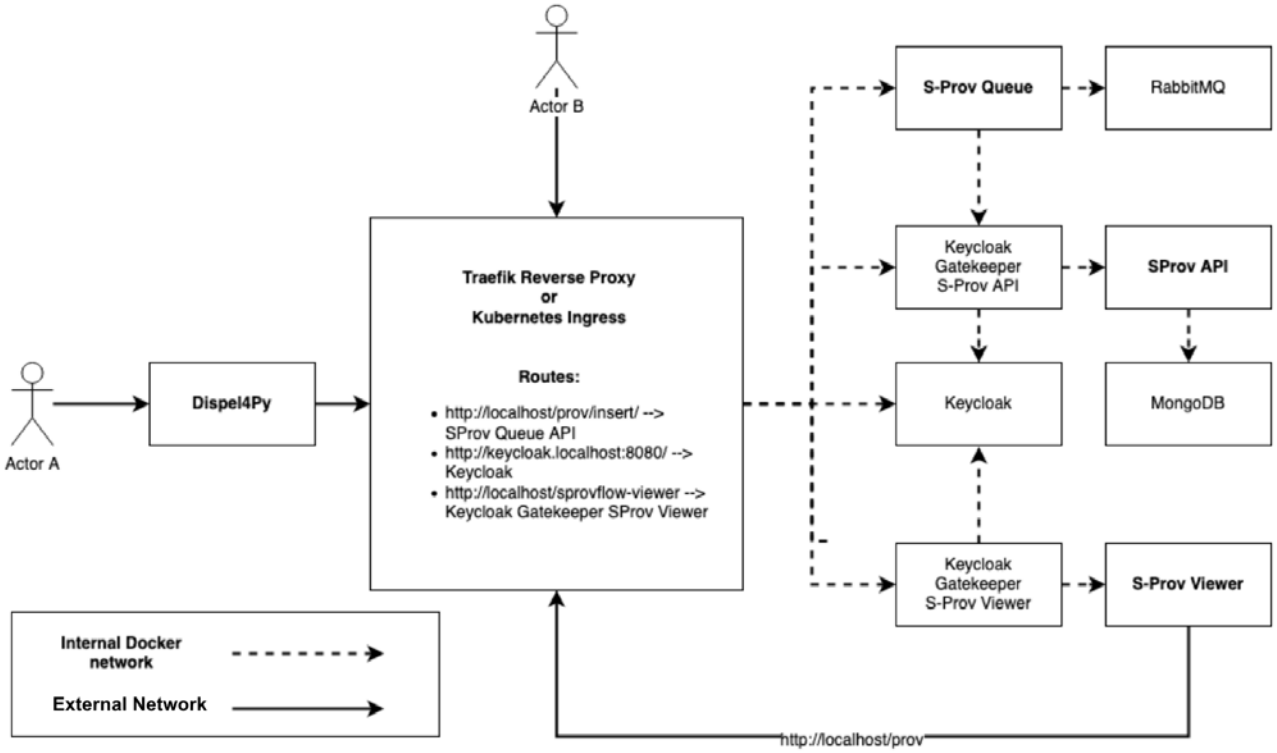

Figure 1: S-ProvFlow integration in DARE. Actor A is the workflow developer interacting with dispel4py, while Actor B accesses the provenance information through the S-ProvFlow viewer. The diagram highlights the communication flows within the Kubernetes cluster, as well as those established from the External Networks, such as the Internet/Intranet.

3 Improved search functionalities

The API allows to perform search for workflow executions and data elements adopting metadata which have been defined by the developer or chosen from standard vocabularies. The updated version accepts now a simple syntax that allows users to formulate queries over multiple terms' single values, ranges or lists, thereby enabling more intuitive and expressive queries than in the past. In the table we report some examples.

Description	**Values Expression**
Comma separated values to express a list **Example:** search by lists of stations' names (seis:station).	AQU,CERA,CAFR
Three dots to express a range **Example:** search by sampling rate's value within a certain range (seis:sampling_rate)	25..30
Single Values **Example:** search by a single channel code (seis:channel)	HXZ

The same combination of terms and values can be used in more API methods (5) (10) (11). Below we show the example of an API call that queries the conjunction of the expression in the table.

```
/workflowexecutions?usernames=<userid>&terms=seis:stations,seis:sampling_rate,seis:channel&expressions=AQU%2CERA%2CAFR,25..30,HXZ&mode=OR&start=0&limit=1000
```

4 Import of CWLProv

Furthermore, DARE is progressing with the adoption of CWL for the implementation of those use cases which benefit from a *task* oriented workflow. This is done incrementally with the refinement of the requirements advanced by WP6 and WP7. Consequently, in order to benefit from the S-ProvFlow system's archiving, advanced metadata queries and tooling, the lineage API has been extended to support the import of provenance information produced by CWL, which is expressed in CWLProv. This is required to map the CWLProv generated by the CWL workflows to S-PROV.

Given the complexity of the CWLProv representation[2], we have initially addressed the provenance output produced by known use cases (eg. WP6 Rapid Assessment and WP7 Cyclone Tracker), aiming eventually at a general solution. More scenarios can be supported, however this will require additional refinements to the acquisition module[3], to make sure the expectations of these and further use cases are met. This will also trigger improvements to the interactive capabilities of the S-ProvFlow system.

5 Testing

The newly developed sets of tests automatically run via the CI/CD pipeline, which is triggered from within gitlab (can be configured to deploy to a target cluster) or manually. These can be divided in two different types of tests, as follows.

5.1 Scoped unit tests

The CI/CD pipeline support, defined in s-ProvFlow[4] executes the unit tests one by one. Thanks to the *coverage*[5] tool all tests are executed automatically. The pipeline is currently configured to run when committing to development branches. This behaviour can be changed to address different testing deployment requirements. In Figure 2 we provide a summary of the current coverage for this type of test.

- Test for CWLProv mapping

[2] CWLProv produced by the execution of a Specfem3d Workflow https://openprovenance.org/store/documents/1977

[3] https://gitlab.com/project-dare/s-ProvFlow/-/blob/master/provenance-api/src/prov-services/cwlprov_2_sprov.py

[4] https://gitlab.com/project-dare/s-ProvFlow/-/tree/master/provenance-api/src/test

[5] http://nedbatchelder.com/code/coverage

- Unit tests for extended query expressions
- Unit tests asserting database queries

Module ↓	statements	missing	excluded	branches	partial	coverage
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/cwlprov_2_sprov.py	363	40	0	206	31	88%
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/flask_raas.py	422	77	0	60	18	79%
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/helper.py	131	33	0	58	8	75%
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/prov/__init__.py	20	12	0	4	0	33%
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/prov/constants.py	92	0	0	8	0	100%
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/prov/identifier.py	75	13	0	8	0	82%
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/prov/model.py	864	427	0	309	30	44%
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/prov/serializers/__init__.py	29	2	0	2	0	94%
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/prov/serializers/provjson.py	188	96	0	98	13	44%
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/prov/serializers/provn.py	15	5	0	2	0	59%
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/prov/serializers/provrdf.py	413	376	0	277	1	6%
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/prov/serializers/provxml.py	190	87	0	110	9	52%
/builds/project-dare/s-ProvFlow/provenance-api/src/prov-services/provenance.py	1391	778	0	626	115	42%
__init__.py	0	0	0	0	0	100%
Total	**4193**	**1946**	**0**	**1768**	**225**	**49%**

coverage.py v5.1, created at 2020-06-02 12:11

Figure 2: Report of the unit tests coverage for the API code.

5.2 API Endpoint tests

These sort of tests run on a local deployment of the API and can be executed manually one by one (e.g. by running the `test_helper.py`) or through the *coverage* toolkit. They simulate calls to the API as they would be sent by the S-ProvFlow *viewer* and assert on *status* code and on the *response object* (returned to the viewer).

6 Conclusions and Future Work

The integration of the lineage services within DARE presented new challenges in respect to the usability, reliability and quality of the whole s-ProvFlow system. Tackling each of these aspects required to re-evaluate design choices, which lead eventually to the updated architecture presented in Section 2. Especially with the implementation of the authentication infrastructure that secures the access to each of the DARE components, we had to rollout solutions to mitigate the impact on the overhead, especially affecting the scientific applications that produce and upload lineage data at runtime. For this reason, our new approach allows the DARE workflows to delegate the authenticated inserts of their lineage traces to the queue. The latter also acts as a resilient messenger, since it can recover in the occurrence of a temporary unavailability of the API.

Improvements to the usability involved the discovery methods of the API, which are now allowing different combinations of metadata query expressions, from lists to value-ranges. This also fostered a better design of the user interface that now exposes these features to end-users. Further work could extend the search capabilities, for instance, enabling free text searches on the metadata values. Such search should be performed across all properties, to assist users in those circumstances where they can not recall the metadata associated with the values of interest.

The underlying model (S-PROV[6]) effectively covered the lineage requirements of DARE. Being the model based on PROV we could import provenance generated by CWL applying a coherent mapping. However, we foresee further extensions. These should address additional metadata linkage across the other registries of the platforms, for instance addressing the source containers and workflow descriptions. These relationships should be automatically established by the platform's API, upon the execution of the workflows. Also, implementing solutions to enrich the metadata recorded by CWLProv, would allow users to gain more benefits from CWL traces, by exploiting the functionalities offered by the lineage services.

Another type of information that could be represented as provenance data consists in asynchronous interaction of the users with the platform, especially when these generate changes to their environment. We have explored these aspects in the framework of other H2020 projects (ENVRIFair, IS-ENES3), parallel to DARE. Here, we develop the SWIRRL API[7], an infrastructure service offering combined and customisable computational environments, such as Notebooks and Visualisation tools. Thanks to the provenance recordings, the system enables different reproducibility actions, such as the restore of an environment to a previous state, or the production of shareable snapshots containing software and data (or means to access it). We believe that the integration of such provenance driven technologies within DARE, would give users further control over the reproducible and trustworthy dissemination of their research.

Annex I S-ProvFlow lineage API

Provenance acquisition

[6] S-PROV - http://purl.org/s-prov-v1-dev

[7] SWIRRL Reproducible Research Labs on the Cloud https://gitlab.com/KNMI-OSS/swirrl/swirrl-api

(1) workflowexecutions/insert	Bulk insert of bundle or lineage documents in JSON format
(2) workflowexecutions/<id>/edit	Update of the description of a workflow execution. Users can improve this information in free text.
(3) workflowexecutions/<id>/delete	Delete a workflow execution trace, including its bundle and all its lineage documents.
(4) workflowexecutions/import	Import lineage traces from other workflow systems and maps them to S-PROV allowing their exploration through the S-ProvFlow tools. The current implementation supports the import of traces in CWLProv.

Table1 : S-ProvFlow API Methods **Provenance acquisition**

Monitoring, validation and lineage queries	
(5) workflowexecutions(/<id> | ?<query string>)	Provides a list of workflow runs performed by one or more users. Runs can be searched by specifying the parameters values used in the workflow and by the metadata associated with the data and the data-formats. Mode of the search can also be indicated (mode ::= (OR | AND). Boolean operators are applied metadata and parameters' values within each run.
(6) workflowexecutions/<id>/showactivity?<query-string>	Extract detailed information of the processes executed in each run. It shows progress, anomalies (such as exceptions or systems' and users messages), count of the data produced and whether it is available for download. This method can also be used for runtime monitoring.
(7) instances/<id> **(8) invocations/<id>** **(9) components/<id>**	Extract details about the invocation of an instance of a workflow process.
(10) data*(/<id> | ?<query string>)*	Extract metadata associated with a *Data* item and its members, the *DataGranules*. The data is selected by specifying the id or a *query-string*. Query parameters allow searching by

	*attribution (*to a workflow or one of its processes/functions) or by specifying metadata *expressions* indicating single values, ranges, lists, and data formats. Mode of the search can also be indicated (mode ::= (OR | AND)).
(11) data/filterByAncestor*?<query string>*	Filter a list of data ids based on the existence of at least one ancestor in their data dependency graph, according to a list of metadata terms and by specifying expressions indicating single values, ranges or lists, and data formats. Maximum depth level and mode of the search can also be indicated.
(12) data/<*id*>/derivedData **(13) data/<*id*>/wasDerivedFrom**	Starting from a specific data entity of the data dependency is possible to navigate through the derived data (11) or backwards across the element's data dependencies (12). The number of traversal steps is provided as a parameter (level).
(14) terms*?<query string>*	Return a list of discoverable metadata terms based on their appearance in a collection of runIds and usernames, passed as parameters, or for the whole provenance archive. Terms are returned indicating their type (when consistently used), min and max values and their number of occurrences within the scope of the search.

Table 2: S-ProvFlow API Methods **Monitoring, validation and lineage queries**

Comprehensive Summaries	
(15) summaries/workflowexecutions/<*id*>*?<query string>*	Returns an overview of the distribution of the computation within a single run. It reports the size of data movements between the workflow components, their instances or invocations depending on the specified granularity level.

(16) summaries/collaborative*?<query string>*	Extract information about the reuse and exchange of data between workflow executions based on terms' values- ranges and a group of users. The API method allows for inclusive or exclusive (mode ::= (OR | AND) queries on the terms' values.

Table 3: S-ProvFlow API Methods **Comprehensive Summaries.** These methods are used to produce visual analytics within the BDV (Bulk Dependencies Visualiser) of the s-ProvFlow. They allow clients to cluster the returning data by specifying a particular property in the *groupBy* parameter.