Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Towards reproducible software studies with MAO and Renku[☆]

Josef Spillner[a,*], Panagiotis Gkikopoulos[a], Pamela Delgado[b], Christine Choirat[b]

[a] *ZHAW School of Engineering, InIT, Technikumstr. 9, PF, 8401 Winterthur, Switzerland*
[b] *SDSC, ETH Zurich and EPFL, Switzerland*

A R T I C L E I N F O

Article history:
Received 9 November 2020
Received in revised form 17 October 2021
Accepted 14 December 2021

Keywords:
Software technology
Data science
Reproducibility

A B S T R A C T

In software engineering, the developers' joy of decomposing and recomposing microservice-based applications has led to an enormous wave of microservice artefact technologies. To understand them better, researchers perform hundreds of experiments and empirical studies on them each year. Improving the reuse and reproducibility of these studies requires two ingredients: A system to automate repetitive experiments, and a research data management system with emphasis on making research reproducible. Both frameworks are now available via the Microservice Artefact Observatory (MAO) and Renku. In this paper, we explain the current capabilities of MAO as a global federated research infrastructure for determining software quality characteristics. Moreover, we emphasise the integration of MAO with Renku to demonstrate how a reproducible end-to-end experiment workflow involving globally distributed research teams looks like.

Code metadata

Current code version	v21.09
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-20-00082
Legal Code License	Apache-2.0
Code versioning system used	git
Software code languages, tools, and services used	python, docker
Compilation requirements, operating environments & dependencies	etcd, postgresql
If available Link to developer documentation/manual	https://github.com/serviceprototypinglab/mao-orchestrator/tree/master/Docs
Support email for questions	pang@zhaw.ch

1. Motivation and significance

Microservice artefacts are basic units of software which are composed to yield executable, distributable and scalable applications. Often appearing as single (atomic) files or compound archives, the characteristics of each of them influences the overall characteristics of the resulting applications at runtime. Software technology researchers are thus interested in code analysis, static property determination at the artefact level, and dynamic runtime assessment at one point in time, as well as in evolution of these metrics over time. From the plethora of already published works, one can estimate the multidimensional space of research on microservices:

1. **Artefact types:** There are conventional artefact types per programming language (for Platform-as-a-Service, or PaaS), such as Java's web application archives (WAR), Python's egg files or Ruby's gems [2], and the respective source files. Often, they are set up as microservices using glue code. There are also directly executable language- and provider-specific cloud function artefacts (for Function-as-a-Service, or FaaS), such as Google Cloud Functions and AWS Lambda, and smart contracts (for Blockchain-as-a-Service, or BCaaS), such as Solidity [3]. Polyglot artefact types further encompass container and virtual machine images (e.g. Docker,

☆ Partially funded by Swissuniversities P-5 Easy FAIR – Supporting the adoption of FAIR and reproducible digital scholarship with Renku (192-010).

* Corresponding author.
E-mail addresses: josef.spillner@zhaw.ch (Josef Spillner), pang@zhaw.ch (Panagiotis Gkikopoulos), pamela.delgado@epfl.ch (Pamela Delgado), christine.choirat@datascience.ch (Christine Choirat).

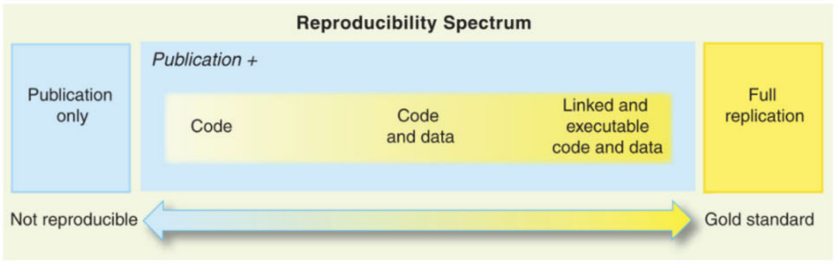


Fig. 1. Reproducibility spectrum.
Source: [1].

OVA) and their build documents (e.g. Dockerfiles [4]). Finally, there are composition documents (e.g. Docker Compose, Kubernetes manifests) and even bundled compositions (e.g. Helm charts).

2. **Artefact repositories:** The spectrum ranges from generic software hosting (e.g. private and public GitLab/GitHub instances) over specialised repositories (e.g. Docker Hub, Maven Central) to hybrid solutions (e.g. Artifactory). Often, there are quality and activity differences between them, making it necessary to use many of them in combination as representative sample.
3. **Lifecycle:** There are works concerned with the design and proper abstraction levels and factors [5], implementation techniques, code transformation, deployment [6], live migration, benchmarking [7] and self-management.
4. **Research methods:** There are empirical studies with interviews [8], quantitative studies based on data collection [9], simulation [10], experiments [11], as well as design science approaches [12].
5. **Research data handling:** Several authors publish no data at all. Others point to datasets, in some cases even reproducible or evolvable ones. They fall into different categories of the reproducibility spectrum. In contrast to other communities, there are no standard repositories for insights into microservices, and few widely accepted reference datasets. Feeding machine-readable data back into development and runtime tools around microservices is however considered useful, as evidenced by the proliferation of knowledge bases [13]. Furthermore, several studies make wide assumptions that can be tightened with representative data samples.

The Microservice Artefact Observatory (MAO) has been designed to overcome the differences and difficulties in automated acquisition, aggregation, verification and analytics of the research data related to microservices. Its main features are federation of independently operated nodes involving several research institutions to established ground truth based on consensus, resilient operation to ensure the data analysis pipeline is working flawlessly, and extensibility for new microservice technologies.

At the single node level, there is apparent overlap with other data management and monitoring frameworks existing in the cloud-native space. However, MAO's main feature is the distributed and collaborative environment it enables. MAO integrates data acquisition tools, but those are meant to be generic tools scheduled for automated operation, therefore differentiating it from monitoring frameworks, such as SonarQube [14]. While MAO has a basic data workflow automation system, similar to frameworks such as Airflow [15], Dagster [16] or Prefect [17], it is primarily a distributed collaboration tool, rather than an automation framework. MAO's primary feature is the ability to share experiment code and configurations between collaborators, and arrange the resulting reproduced data in a way that it can easily be compared and cross-verified and to partially automate this verification.

Once all data has been aggregated and confirmed, the scope of MAO ends. In this article, after presenting MAO itself, we thus demonstrate how to complement it with Renku. As opposed to static data repositories, Renku serves not only as one-time sink but rather as long-term platform for other researchers to curate, augment and engage with the data continuously injected by MAO for increased reproducibility (see Fig. 1).

2. Software description

2.1. Microservice Artefact Observatory

The task of the observatory is to gather information and insights on software artefacts in the form of metrics which allow assessing entire software applications in which these artefacts are used as microservices. The metrics may refer to completeness of metadata, code quality, presence of security vulnerability, unit test coverage, startup latency, robustness in the presence of fuzz testing or elasticity, among other characteristics. The specifics of these metrics are user-defined. To convert an observation tool for deployment to MAO, it only needs to adhere to a specific folder structure and be accompanied with a simple metadata file that acts as a specification for the data the tool produces. As with all automated solutions, the long-term aim is to remove the need to conduct isolated and error-prone data acquisition for each study. Instead, researchers can just query a set of metrics over a set of artefact repositories within a time window, and base their analysis on the query result which is guaranteed to be stable, verified and citable.

The observatory consists of an generic underlying federated infrastructure in which nodes are operated independently and coordinated via consensus voting. Nodes fetch new data and/or create new data through aggregation and analytics. All operations are meant to be resilient, automatable and auditable in the interest of providing a ground truth knowledge about the software artefacts with the highest possible level of verification from trusted operators. This knowledge is then suitable to be cited and used for further studies by researchers. Specifically, MAO aims at these factors:

1. **Automation:** Repetitive data acquisition and exploration should be repeated with fixed time periods to yield a precise representation. A task scheduler is therefore a key component, while still allowing ad-hoc invocation for early experiments.
2. **Resilience:** In case a network connection fails or a scheduled task produces no or evidently unusable results, it should be marked for repetition within a tolerance window after the originally scheduled time. To tolerate more

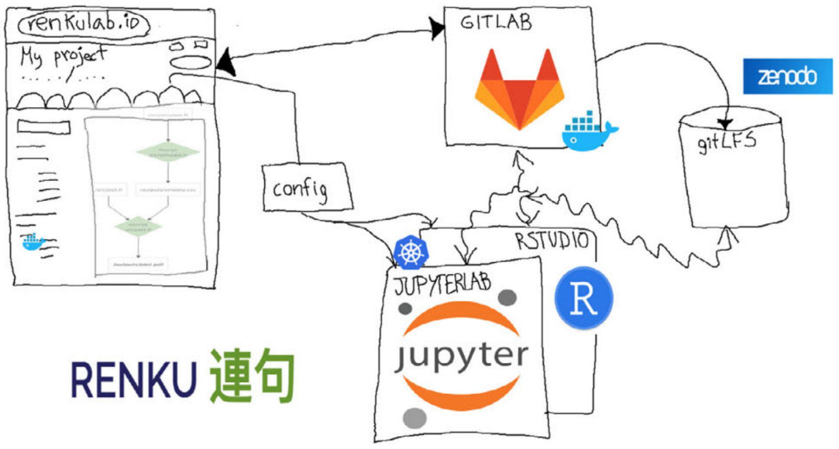


Fig. 2. The Renku Platform.
Source: [18].

extended failure scenarios, redundancy can be achieve by scheduling an experiment on multiple nodes.

3. **Voting:** If there are gaps or conflicts in the output data of nodes that are running the same experiment, a conflict resolution system based on simple voting algorithms will attempt to provide a solution that acts as the ground truth. Researchers can specify which algorithm they wish to be used for this in their metric specification file. At this stage the prototype implementation supports majority voting, weighted mean voting or the option to list all diverging values instead of resolving the conflict automatically.
4. **Audits:** Information about when which data were acquired or which experiments were conducted is logged along with hardware information about the node so that a comparison of for instance different runtime behaviour can be further investigated.

A number of observatory modules have been implemented specifically to address microservice artefacts. Thus, each MAO node is capable to schedule a daily retrieval of metadata from Docker Hub or a weekly download of serverless applications from the AWS Serverless Application Repository, among other types of artefacts. The determination of aggregate values, such as the average number of maintainers per artefact, is part of these modules but can also be implemented as dedicated aggregation modules that are scheduled in succession. Moreover, MAO ensures that all metrics are kept in a timeline so that the evolution of the software artefacts as well as developer/maintainter activities can be tracked over time.

MAO is primarily driven by Zurich University of Applied Sciences, although with increasing participation of interested researchers from other institutions, as evidenced on its website.[1] Available publications cover the framework itself [19,20] as well as artefact type-specific quantitative assessments, covering Helm charts [21], SAM applications [22], DApps [23], Docker images and Dockerfiles [24].

2.2. Renku

The Renku platform consists of RenkuLab, a web-based application and Renku, a command-line tool for managing code, data, workflows and making practical use of the knowledge graph. It focuses on three pillars: reproducibility, reusability and collaboration. A public instance of RenkuLab is available[2] but institutions can also host their own instances and federate them.

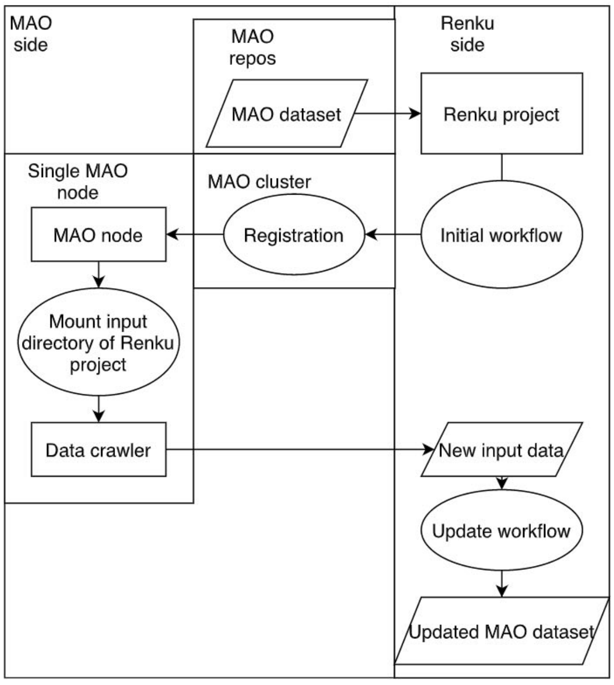


Fig. 3. Integrated Workflow.

Renku workspaces are divided into projects, which are further linked with datasets and interactive environments as well as an integrated GitLab instance in which the Renku project is persisted as versioned GitLab project (Fig. 2). Environments such as R Studio, Jupyter Notebooks and interactive shells are available.

The backbone of Renku is the knowledge graph, used to record and query the relationships between data and code. The analysis workflows are linked with a knowledge graph in the following way: 1. Inputs and outputs of analysis steps are recorded into a knowledge graph while the work is being done. 2. These steps can

[1] MAO website: https://mao-mao-research.github.io/.

[2] RenkuLab website: https://renkulab.io/.

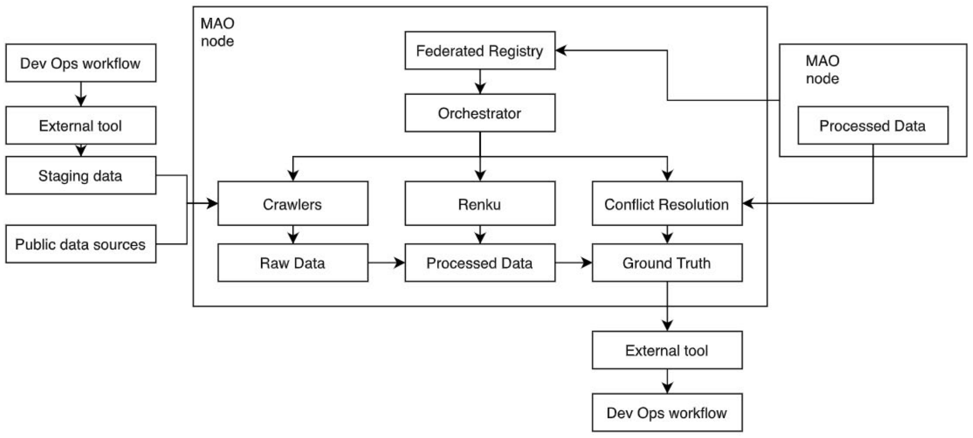


Fig. 4. Data Pipeline.

be repeated or integrated into more complex workflows. 3. The provenance of all data products is always accessible via simple tools. 4. Version control is built-in for data, code and workflows.

2.3. Integrated system

There are two alternative designs to integrate MAO with Renku: Running MAO as interactive environment within Renku, or using Renku's API to store the output data of MAO and perform data aggregation and analysis as a reproducible Renku workflow. We have implemented the latter design due to the loose coupling advantage. MAO's interfacing with a regular Git repository is substituted by interfacing with the repository provided by the configured Renku project. In addition, the MAO orchestrator will use Renku's command-line interface to run the reproducible workflow (Fig. 3) and update the output datasets.

3. Illustrative example

We demonstrate the capabilities of MAO with an end-to-end data analysis pipeline. We start with a standard pipeline including data acquisition, aggregation and analysis. Then we outline the conversion of said pipeline to take advantage of the automation features of MAO and Renku.

3.1. Standard pipeline

The standard pipeline for this example (Fig. 4) will include the following stages:

1. **Data acquisition:** Execution of a tool that gathers raw data, such as a downloader for software artefacts and code repositories or a web scraper for software metadata. This can be scheduled periodically with the use of cron jobs or similar tools.
2. **Aggregation:** Once enough data is collected, a user-provided aggregation script will generate a dataset from the raw data, converting them to a useable format for analysis.
3. **Analysis:** The dataset can now be used for analysis in various ways, for example with R scripts or Jupyter notebooks.

3.2. Converting to a Renku project

The first step to convert this pipeline is to setup a Renku project. The raw data can be used as input data, and the aggregation and analysis scripts can be added to the project. Then, the aggregation and analysis steps of the pipeline can be recorded as a Renku reproducible workflow. This will allow the dataset to be regenerated and the analysis to be updated, whenever new input data is pushed to the repository.

3.3. Data acquisition with the MAO orchestrator

The researchers will need to setup a MAO orchestrator instance. While the system can be started as standalone via Docker-Compose, it can also be configured to join an existing MAO federation, in which case it will have access to the shared registry of data acquisition tools (corresponding to stage 1 of the pipeline above) and datasets of the federation (corresponding to Renku projects).

The data acquisition tool will need to be containerised so that the orchestrator can interact with it via the Docker API. After that the tool can be registered into the orchestrator which will add it to the registry, so that other members of the federation can deploy it as well. The new tool can now be scheduled with standard crontab syntax to run periodically via the orchestrator interface.

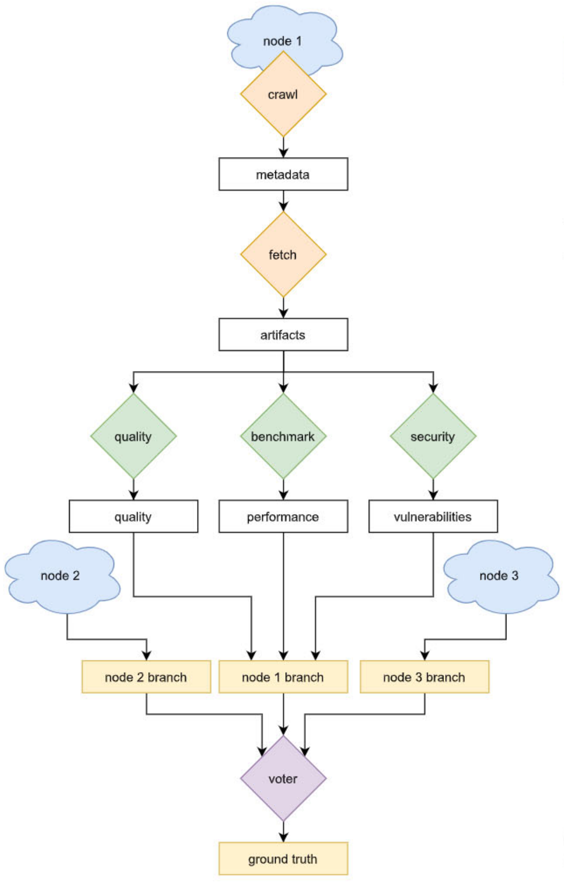
3.4. Renku reproducible workflow

Once the data acquisition is completed, the orchestrator will update the Renku repository with the new input data. Then it can automatically invoke Renku's update functionality and run the previously recorded reproducible workflow, updating both the aggregated dataset and the analysis results.

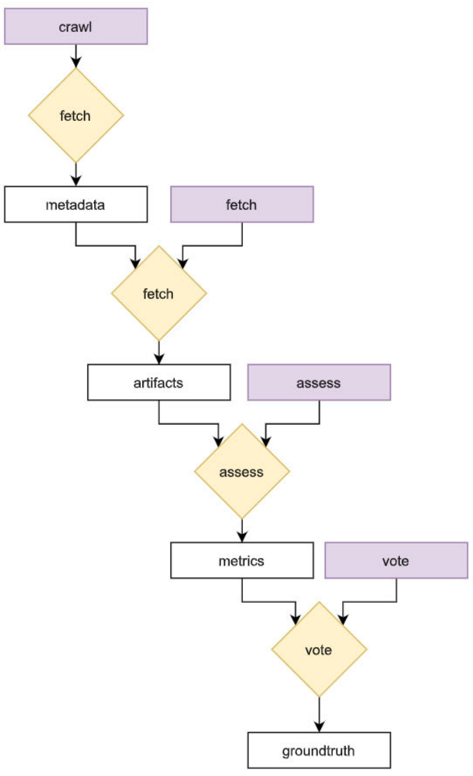
This means that with every scheduled execution of the data acquisition tool, the entire pipeline will run automatically, from the raw data to the analysis results. The knowledge graph (Fig. 5) is updated on each run.

4. Impact

While our research around MAO maintains the focus on microservices, the underlying framework could also modernise quantitative research on other discrete data points retrieved through the network in regular intervals. This encompasses finance data (e.g. foreign exchange rates or stock tickers), health



(a) Conceptual diagram of the integration



(b) Knowledge graph generated by Renku (schematic reproduction)

Fig. 5. Visualisation of the MAO-Renku integrated workflow.

data (e.g. pandemic cases per administrative region), weather and social networks. We expect that beyond the current focus of funding agencies on the existence of a data management plan, increasingly the emphasis will shift towards highly capable research infrastructures and research data management systems. With MAO and its option to interface with Renku, this concern can be addressed. MAO's containerised tool sharing feature allows data acquisition tools to be reproducible, and experiments to thus be repeated and verified with minimal setup. Renku, on the other hand can handle the data management in a reproducible manner and automate the analysis, granting easy access to replication and verification studies, as well as collaborations. For the software example we discussed above, MAO can automate the acquisition and also allow anyone to reproduce it, while Renku can automate the process of repeating the analysis when new data is acquired, while maintaining data provenance and versioning. In the collaborative example, different institutions can pool aggregate data using MAO from their own monitoring infrastructures (e.g. software defects, weather or pandemic related data) and easily share a Renku workflow to manage and analyse it in the same way.

5. Conclusions

MAO delivers a federated infrastructure to perform independent quantitative software metrics observations, to foster a shared ground truth by comparing the observations, and to gain insights through reproducible experiments. The reproducibility is ensured by Renku, the usage of which is transparently integrated into MAO for this matter. Software engineering researchers looking for appropriate tools to perform long-term observations, evolution/trend studies and assessments of recently emerging artefact technologies should consider the use of MAO to benefit from these characteristics.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Peng RD. Reproducible research in computational science. Science 2011;334(6060):1226–7.
[2] Cleare J, Iacob C. Gemchecker: Reporting on the status of gems in ruby on rails projects. In: 2018 IEEE international conference on software maintenance and evolution, ICSME 2018, Madrid, Spain, September (2018) 23-29. IEEE Computer Society; 2018, p. 700–4. http://dx.doi.org/10.1109/ICSME.2018.00080.
[3] Chapman P, Xu D, Deng L, Xiong Y. Deviant: A mutation testing tool for solidity smart contracts. In: IEEE international conference on blockchain, blockchain 2019, Atlanta, GA, USA, July (2019) 14-17. IEEE; 2019, p. 319–24. http://dx.doi.org/10.1109/Blockchain.2019.00050.
[4] Oumaziz MA, Falleri J, Blanc X, Bissyandé TF, Klein J. Handling duplicates in dockerfiles families: Learning from experts. In: 2019 IEEE international conference on software maintenance and evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019. IEEE; 2019, p. 524–35. http://dx.doi.org/10.1109/ICSME.2019.00086.
[5] Al-Debagy O, Martinek P. A metrics framework for evaluating microservices architecture designs. J Web Eng 2020;19(3–4):341–70. http://dx.doi.org/10.13052/jwe1540-9589.19341.

[6] Bravetti M, Giallorenzo S, Mauro J, Talevi I, Zavattaro G. A formal approach to microservice architecture deployment. In: Bucchiarone A, Dragoni N, Dustdar S, Lago P, Mazzara M, Rivera V, Sadovykh A, editors. Microservices, science and engineering. Springer; 2020, p. 183–208. http://dx.doi.org/10.1007/978-3-030-31646-4_8.

[7] Gan Y, Zhang Y, Cheng D, Shetty A, Rathi P, Katarki N, Bruno A, Hu J, Ritchken B, Jackson B, Hu K, Pancholi M, He Y, Clancy B, Colen C, Wen F, Leung C, Wang S, Zaruvinsky L, Espinosa M, Lin R, Liu Z, Padilla J, Delimitrou C. Unveiling the hardware and software implications of microservices in cloud and edge systems. IEEE Micro 2020;40(3):10–9. http://dx.doi.org/10.1109/MM.2020.2985960.

[8] Haselbock S, Weinreich R, Buchgeher G. An expert interview study on areas of microservice design. In: 11th IEEE conference on service-oriented computing and applications, SOCA 2018, Paris, France, November (2018) 20-22. IEEE Computer Society; 2018, p. 137–44. http://dx.doi.org/10.1109/SOCA.2018.00028.

[9] Wist K, Helsem M, Gligoroski D. Vulnerability analysis of 2500 docker hub images. 2020, CoRR abs/2006.02932, arXiv:2006.02932, https://arxiv.org/abs/2006.02932.

[10] Shadija D, Rezai M, Hill R. Microservices: Granularity vs. performance. In: Anjum A, Sill A, Fox GC, Chen Y, editors. Companion proceedings of the 10th international conference on utility and cloud computing, UCC 2017, Austin, TX, USA, December (2017) 5-8. ACM; 2017, p. 215–20. http://dx.doi.org/10.1145/3147234.3148093.

[11] Malawski M, Gajek A, Zima A, Balis B, Figiela K. Serverless execution of scientific workflows: Experiments with hyperflow, AWS lambda and google cloud functions. Future Gener Comput Syst 2020;110:502–14. http://dx.doi.org/10.1016/j.future.2017.10.029.

[12] Yu D, Jin Y, Zhang Y, Zheng X. A survey on security issues in services communication of microservices-enabled fog applications. Concurr Comput Pract Exp 2019;31(22). http://dx.doi.org/10.1002/cpe.4436.

[13] Apel S, Hertrampf F, Späthe S. Toward a knowledge model focusing on microservices and cloud computing. Concurr Comput Pract Exp 2020;32(13). http://dx.doi.org/10.1002/cpe.5414.

[14] Joshi SL, Deshpande B, Punnekkat S. Experimental analysis of dependency factors of software product reliability using sonarqube. In: Tarhan AK, Coskunçay A, editors. Joint proceedings of the international workshop on software measurement and the international conference on software process and product measurement (IWSM Mensura 2019), Haarlem, the Netherlands, October (2019) 7-9, 2476 of CEUR Workshop Proceedings. CEUR-WS.org; 2019, p. 130–7, http://ceur-ws.org/Vol-2476/short5.pdf.

[15] Singh P. Airflow. Berkeley, CA: A Press; 2019, p. 67–84.

[16] Dagster. 2021, https://dagster.io/, accessed: 2021-10-07.

[17] Prefect. 2021, https://www.prefect.io/, accessed: 2021-10-07.

[18] Jablonski ER. The renku platform. 2020, online: https://renkulab.io/gitlab/team-renku/easyfair/-/blob/master/imgs/renku_component_ecosystem.png.

[19] Gkikopoulos P. Data distribution and exploitation in a global microservice artefact observatory. In: 15th IEEE world congress on services (SERVICES), Milan, Italy. 2019, p. 319–22. http://dx.doi.org/10.1109/SERVICES.2019.00089.

[20] Gkikopoulos P, Spillner J, Schiavoni V. Monitoring data distribution and exploitation in a global-scale microservice artefact observatory. 2020, arXiv:2006.01514.

[21] Spillner J. Quality assessment and improvement of helm charts for kubernetes-based cloud applications. 2019, arXiv:1901.00644.

[22] Spillner J. Quantitative analysis of cloud function evolution in the AWS serverless application repository. 2019, arXiv:1905.04800.

[23] Qasse IA, Spillner J, Talib MA, Nasir Q. A Study on DApps Characteristics, in: 2nd IEEE international conference on decentralized applications and infrastructures, 2020, to appear - conference shifted due to coronavirus.

[24] Müller M, Rüdisüli M. DQA: Docker quality analysis. 2020, Docker image: https://hub.docker.com/r/svl7/dqa-mao.